
Bayes, AI and Deep Learning

Foundations of Data Science

Nick Polson and Vadim Sokolov



Table of contents

Preface	5
Preface	5
The Modern AI Playbook	7
The Modern AI Playbook	7
Generative AI	9
Levels of Automation: AIQ, AGI, ASI	11
Software Development: The First Frontier for Agents	11
AI Agents: From Prediction to Action	12
Physical AI: Embodied Intelligence	12
The Human Dimension: Dignity and Economics	13
Large Language Models (LLMs)	16
Business Models and Market Impact	18
Bayes: Evidence as Negative Log-Probability	18
From Beliefs to Decisions: Expected Utility	22
Statistical Learning: Patterns in Data	23
Examples: AI in Action	24
Anything as a Vector	28
Tokenization and Embeddings	28
Vectorizing Chess	29
Human vs. Machine Representations	30
How Machines Learn: The Loss Function	30
Book Structure	31
Course Plans	32

I Bayes	33
1 Probability and Uncertainty	35
1.1 Odds as Probabilities	39
1.2 Random Variables: Quantifying Uncertainty	47
1.2.1 Continuous Random Variables	49
1.2.2 The Inverse CDF Method	51
1.2.3 Functional Transformations	52
1.3 Expectation and Variance (Reward and Risk)	53
1.3.1 Standard Deviation and Covariance	54
1.3.2 Portfolios: linear combinations	55
1.4 Limiting Behavior of Averages	57
1.4.1 The Weak Law of Large Numbers	58
1.4.2 Kolmogorov's Strong Law of Large Numbers	58
1.5 Binomial, Poisson, and Normal Distributions	60
1.5.1 Bernoulli Distribution	61
1.5.2 Poisson Distribution	63
1.5.3 Normal Distribution	65
1.6 Conditional, Marginal and Joint Distributions	67
1.7 Independence	69
1.8 Further Notes: Dutch Book Arguments	70
2 Bayes Rule	73
2.1 Law of Total Probability	74
2.2 Intuition and Simple Examples	75
2.3 Real World Bayes	80
2.4 First Application: Naive Bayes	91
2.5 Sensitivity and Specificity	95
2.5.1 Medical Diagnostics	97
2.6 Advanced Applications	100
2.7 Graphical Representation of Conditional Independence.	103

<i>Contents</i>	5
3 Bayesian Learning	107
3.1 Exchangeability and the Bayesian view of probability models	109
3.1.1 de Finetti's representation theorem	112
3.1.2 Posterior Empirical CDF	114
3.2 Sufficient Statistic	116
3.3 Beta-Binomial Model	118
3.4 Poisson Model for Count Data	124
3.5 Poisson-Gamma: Learning about an Intensity	129
3.6 Exponential-Gamma Model	132
3.7 Normal With Unknown Mean	136
3.7.1 Posterior Predictive	140
3.8 Normal With Unknown Variance	140
3.8.1 The Normal-Gamma Model	141
3.8.2 Credible Intervals for Normal-Gamma Model Posterior Parameters	142
3.9 Multivariate Normal	142
3.10 Mixtures of Conjugate Priors	146
3.11 Bayesian Computation: when conjugacy breaks	149
3.11.1 Posterior Consistency and the Law of Large Numbers	149
3.11.2 Monte Carlo Methods	150
3.11.3 Markov Chain Monte Carlo	151
4 Utility, Risk and Decisions	155
4.1 Expected Utility	156
4.2 Historical Perspectives on Optimal Decision Making	166
4.2.1 De Finetti's Insurance Problem	166
4.2.2 Markowitz's Recognition	167
4.2.3 Dynamic Kelly and Bellman's Principle	168
4.2.4 Livermore's Trading Wisdom	168
4.3 Statistical Decisions and Risk	170
4.4 Unintuitive Nature of Decision Making	172

4.5 Decision Trees	181
4.6 Nash Equilibrium	187
5 A/B Testing	193
5.1 Hypothesis Testing	194
5.2 Confidence Intervals	199
5.2.1 Mythbusters Example	200
5.2.2 Other Applications	204
5.3 A/B Testing Applications	208
5.4 Challenges in A/B Testing	213
5.4.1 Multiple Testing	213
5.5 Randomized Controlled Trials and Causality	214
5.5.1 The Question of Causation	217
5.5.2 Split-plot designs and Rothamsted	223
5.5.3 Digital experimentation (A/B testing)	224
5.5.4 The era of observational data	225
5.5.5 From experiments to observational data	225
6 Bayesian Hypothesis Testing	227
6.1 Likelihood Principle	229
6.2 The Bayesian Approach	231
6.3 Interval Estimation: Credible Sets	240
6.4 Alternative Approaches	243
6.4.1 Significance testing using p-values	243
6.4.2 Bayes vs P-value	245
6.4.3 Neyman-Pearson	250
6.5 Sequential Analysis	251
6.5.1 The Bayesian Framework for Sequential Testing	251
6.5.2 Wald's Sequential Probability Ratio Test	252
6.5.3 Applications in Clinical Trials	252
6.5.4 Sequential Analysis for Rare Diseases	258

<i>Contents</i>	7
6.5.5 Multi-Armed Bandit Experiments	259
6.5.6 Practical Considerations	263
6.6 Examples and Paradoxes	264
6.7 Prior Sensitivity	267
6.8 The difference between p-values and Bayesian evidence	269
6.9 Jeffreys' Decision Rule	270
6.10 Cromwell's Rule	272
7 Stochastic Processes	273
7.1 Brownian Motion	274
7.1.1 Optimal Portfolio Allocation: The Merton Rule	278
7.2 Black-Scholes Model for Sports Betting	279
7.2.1 Implied Volatility for Sports Games	280
7.3 Poisson Process	289
7.4 The Lévy-Itô Decomposition in Finance	297
7.5 Newton and the South Sea Bubble	299
7.6 Stochastic Volatility: Financial Economics	301
7.6.1 Motivation: Combining Brownian Motion and Jumps	302
7.6.2 The Stochastic Volatility Model	303
7.6.3 Bayesian Inference for Stochastic Volatility Models	307
8 Gaussian Processes	311
8.1 Making Predictions with Gaussian Processes	313
9 Reinforcement Learning	321
9.1 Multi-Armed Bandits	322
9.1.1 When to End Experiments	323
9.1.2 Contextual Bandits	327
9.1.3 Summary of MAB Experimentation	328
9.2 Bellman Principle of Optimality	329
9.3 Markov Decision Processes	332
9.3.1 Mathematical Representation	333

9.3.2 MDP Solvers	343
9.3.3 Model-Free Methods	346
9.4 Q-Learning	349
9.5 Bayesian Optimization	350
9.6 Concluding Remarks	355
II AI	357
10 Unreasonable Effectiveness of Data	359
10.1 The Wisdom and Madness of Crowds	366
10.1.1 Historical Delusions and Economic Bubbles	366
10.1.2 Galton's Ox: The Wisdom of Aggregation	368
10.1.3 Smart Money, Dumb Money: Learning from Crowds . .	369
11 Pattern Matching	375
11.1 Why Pattern Matching?	375
11.1.1 Richard Feynman on Pattern Matching and Chess . .	376
11.2 Supervised Learning	377
11.3 Complex Functions	382
11.4 Model Estimation	383
11.4.1 Penalized Likelihood	385
11.4.2 Bayesian Approach	387
11.5 Prediction Accuracy	387
11.5.1 Evaluation Metrics for Regression	389
11.5.2 Evaluation Metrics for Classification	390
12 Linear Regression	393
12.1 Statistical Properties of Linear Models	400
12.1.1 Estimates and Intervals	401
12.2 Factor Regression and Feature Engineering	412
12.2.1 Logarithmic and Power Transformations	412
12.3 Interactions	422

<i>Contents</i>	9
12.4 Categorical Variables and Dummy Encoding	430
12.5 Quantile Regression	437
12.6 Bayes Regression	439
12.6.1 Penalized Regression and Priors	442
12.6.2 Global-Local Priors and the Horseshoe	442
13 Logistic Regression and Generalized Linear Models	445
13.1 Model Fitting	446
13.2 Confusion Matrix	451
13.3 ROC Curve and Confounding Variables	457
13.4 Imbalanced Data	464
13.5 Kernel Trick	465
13.6 Generalized linear models	468
13.6.1 Deviance and model comparison	469
13.7 Bayesian Logistic Regression with The Polya-Gamma Distribution	471
13.7.1 The Data-Augmentation Strategy	472
13.8 Bayesian Analysis of Horse Race Betting	474
13.8.1 Theoretical Guarantees	475
14 Tree Models	479
14.1 Building a Tree via Recursive Binary Splitting	484
14.2 Pruning: Taming an Overfit Tree	485
14.2.1 Example: Boston Housing Data	486
14.3 Classification Trees	489
14.3.1 Example: Classifying Diamond Quality	490
14.4 Ensemble Methods	491
14.4.1 Bagging	492
14.4.2 Random Forest	493
14.4.3 Boosting	494
14.5 BART for causal inference	497
14.5.1 BART versus Propensity Score Matching (PSM)	506

14.5.2 Conclusion	508
14.6 Appendix: Theoretical Foundations	509
14.6.1 Why Ensembles Work: A Geometric Perspective	509
14.6.2 Smoothing and Variance Reduction	509
14.6.3 The Problem of High Dimensions	509
14.6.4 Trees as “Adaptive” Nearest Neighbors	510
14.7 Classification variance decomposition	512
14.7.1 The Nearest Neighbor Insight	514
15 Forecasting	517
15.0.1 Epidemic Forecasting: A Motivating Example	518
15.0.2 The bsts Package	518
15.1 Structural time series models	519
15.2 Regression with spike and slab priors	525
15.3 Model diagnostics: did the Google data help?	527
15.4 Final Remarks on Structural Models	534
15.5 Beyond State-Space Models: Machine Learning Approaches	535
15.5.1 Modern Era Forecasting	536
15.6 Quantile Regression Forests.	538
15.7 Deep Learning for Time Series: Temporal Fusion Transformers	539
15.7.1 Benchmark: BSTS vs. TFT on Initial Claims	540
15.8 Conclusion	542
15.9 Advanced Theory: Under the Hood	543
15.9.1 Kalman Filtering	544
15.9.2 HMM: Hidden Markov Models	553
15.9.3 Mixture Kalman filter	557
15.9.4 Regime Switching Models	558
15.10 Particle Learning for General Mixture Models	560
15.10.1 The Particle Learning Algorithm	561
16 Model Selection	565
16.1 Fundamental Considerations in Model Selection	565

<i>Contents</i>	11
16.1.1 Model Complexity and Generalization	565
16.1.2 Overfitting and Underfitting	566
16.1.3 Data Quality and Quantity	566
16.1.4 Model Explainability	567
16.1.5 Computational Cost	567
16.1.6 Ethical Considerations	567
16.2 Prediction vs Interpretation	568
16.2.1 Breiman’s Two Cultures	569
16.3 Diagnostics for Model Assumptions	570
16.4 Out-of-Sample Performance	574
16.5 Bias-Variance Trade-off	577
16.6 Cross-Validation	579
16.7 Model evaluation: calibration, discrimination, and scoring	581
16.7.1 Calibration and calibration plots	581
16.7.2 Proper scoring rules (and the decision-theory link)	581
16.8 Bayesian Model Selection	582
16.8.1 Bayesian Information Criterion (BIC)	588
16.8.2 Neural Information Criterion (NIC)	590
16.8.3 Automatic Relevance Determination (ARD)	591
16.9 Model Selection and Bayesian Relativity	596
16.9.1 Exhaustive vs Non-Exhaustive Hypotheses	596
16.10 The Asymptotic Carrier	597
16.11 Model Explainability	598
16.11.1 The Imperative for Explainability	599
16.11.2 The Paradox of Understanding and Utility	600
16.12 Model Elaboration and Nested Model Testing	602
16.12.1 The Dickey-Savage Approach to Nested Models	603
16.13 Conclusion	604
17 Statistical Learning Theory and Regularization	607
17.1 Normal Means Problem	608

17.2 Unbiasedness and the Optimality of Bayes Rules	614
17.2.1 Full Bayes Shrinkage	618
17.3 Bias-Variance Decomposition	619
17.3.1 Risk Decomposition	620
17.4 Sparsity	621
17.5 Maximum A posteriori Estimation (MAP) and Regularization .	622
17.6 The Duality Between Regularization and Priors	624
17.6.1 MAP as a Poor Man's Bayesian	625
17.7 Ridge Regression (ℓ_2 Norm)	626
17.7.1 Tikhonov Regularization Framework	626
17.7.2 Kernel View of Ridge Regression	630
17.8 Scale Mixtures Representations	631
17.9 Lasso Regression (ℓ_1 Norm)	631
17.9.1 Lasso as a Scale Mixture	636
17.9.2 More Mixture Representations	637
17.10 Horseshoe	638
17.10.1 Mathematical Formulation	638
17.10.2 Shrinkage Properties	639
17.10.3 Comparison with Other Priors	640
17.10.4 Computational Implementation	640
17.10.5 Comparison and Package Implementation	643
17.11 Bridge (ℓ_α)	645
17.11.1 Bayesian Framework and Data Augmentation	645
17.12 Full Bayes for Sparsity Shrinkage	647
17.12.1 Spike-and-Slab Prior	647
17.13 Subset Selection (ℓ_0 Norm)	650
17.13.1 Connection to Spike-and-Slab Priors	650
17.13.2 Single Best Replacement (SBR) Algorithm	651
17.14 Advanced Topics in Regularization	657
17.14.1 The Vertical Likelihood Duality	657
17.14.2 Fundamental Integral Identities	658

<i>Contents</i>	13
17.14.3 Quantile Regression Example	658
17.14.4 Improper Limit Cases	658
17.14.5 Computational Advantages of Scale Mixtures	659
17.14.6 Generalized Ridge Regression in the Canonical Basis	659
17.14.7 Trend Filtering and Composite Penalties	660
17.15 Final Thoughts	660
III Deep Learning	667
18 Neural Networks	669
18.1 Introduction	670
18.2 Mathematical Foundations	672
18.2.1 ReLU Networks as Max-Sum Networks	672
18.3 Classification with Neural Networks	673
18.4 Activation Functions	677
18.5 Unsupervised Learning: Auto-Encoders	679
18.5.1 Dynamic Auto-Encoders	680
18.6 Modern Deep Learning Architectures	680
18.6.1 Transfer Learning and Pre-trained Blocks	681
18.6.2 Mixture of Experts	682
18.6.3 Ensembles	683
18.6.4 Architectural Innovations	685
18.7 Summary	685
19 Theory of Deep Learning	687
19.1 Ridge and Projection Pursuit Regression	687
19.1.1 From Approximation to Representation	688
19.2 Kolmogorov Superposition Theorem (KST)	689
19.2.1 Kolmogorov-Arnold Networks	690
19.3 Kolmogorov Generalized Additive Models (K-GAM)	690
19.4 Space Partitioning	692

19.4.1 Connection to Bayesian Model Averaging	694
19.5 Kernel Smoothing and Interpolation	694
19.6 Transformers as Kernel Smoothing	696
19.6.1 Transformer	696
19.7 Deep Learning as Representation Learning	698
19.7.1 Dimensionality Expansion vs. Reduction	699
19.7.2 Linear Baselines: PCA and PLS	699
19.7.3 Uncertainty Quantification	700
19.8 Double Descent	700
19.9 Application	704
19.9.1 Simulated Data	704
19.9.2 Training Rates	705
19.10 Conclusion	707
20 Gradient Descent	709
20.1 Deep Learning and Least Squares	711
20.2 Stochastic Gradient Descent	716
20.3 Automatic Differentiation (Backpropagation)	718
20.4 Advanced Optimization Methods	725
20.4.1 Momentum Methods	725
20.4.2 Adaptive Learning Rate Methods	726
20.4.3 Second-Order Methods	727
20.5 Why Robbins-Monro?	727
20.5.1 Connection to M-Estimation	728
20.6 The EM, ECM, and ECME Algorithms	729
20.6.1 ECM and ECME Extensions	731
20.7 Conclusion	733
21 Quantile Neural Networks	735
21.1 Quantile Regression	736
21.2 From Densities to Quantiles: A Generative Approach	738

<i>Contents</i>	15
21.3 Nonlinear Quantile Regression via Trend Filtering	740
21.4 Bayes Rule for Quantiles	743
21.5 Maximum Expected Utility via Quantile Neural Networks . . .	744
21.5.1 Implementation Strategy	745
21.6 Neural Network Implementation	747
21.6.1 Learning Multiple Quantiles Simultaneously	748
21.6.2 Non-Crossing Constraints	749
21.6.3 Cosine Embedding for τ	749
21.6.4 Synthetic Data Example	750
21.7 Portfolio Optimization with Quantile Neural Networks	752
21.7.1 Empirical Example	753
21.8 Supply Chain Forecasting at Scale	754
21.8.1 Demand Forecasting Setup	755
21.8.2 Implementation Strategy	757
21.9 Distributional Reinforcement Learning	759
21.10 Discussion and Summary	760
22 Convolutional Neural Networks	763
22.1 The MNIST Dataset	763
22.2 Fully Connected Approach	764
22.3 Convolutions	766
22.3.1 Key Concepts	766
22.3.2 A Toy Example	767
22.4 CNN for MNIST	769
22.5 Summary	770
23 Natural Language Processing	771
23.1 Converting Words to Numbers (Embeddings)	771
23.1.1 The Math of Twenty Questions	772
23.2 Word2Vec and Distributional Semantics	775
23.3 Word2Vec for War and Peace	775

23.3.1 The Skip-Gram Model	777
23.3.2 The Continuous Bag of Words (CBOW) Model	778
23.3.3 Pretraining Word2Vec	779
23.4 Computational Efficiency Through Negative Sampling	781
23.5 Global Vectors and Matrix Factorization	782
23.6 Beyond Words: Subword and Character Models (Tokenization)	783
23.7 Attention Mechanisms and Contextual Representations	784
23.8 Kernel Smoothing as Attention	787
23.8.1 Attention over a sequence of words	789
23.9 Cross-attention for Translation	792
23.9.1 Multi-Head Attention: Parallel Perspectives	793
23.9.2 Positional Information in Attention	794
23.9.3 Computational Efficiency and Practical Considerations .	795
23.9.4 From Attention to Modern Language Models	795
23.10 Transformer Architecture	796
23.10.1 Computational Complexity and Scalability	799
23.11 Pretraining at Scale: BERT and Beyond	800
23.11.1 BERT Architecture and Training Details	800
23.11.2 Data Preparation for BERT Pretraining	802
23.12 Transfer Learning and Downstream Applications	803
23.13 Model Compression and Efficiency	804
23.14 Theoretical Perspectives and Future Directions	804
23.15 Conclusion	805
24 Large Language Models	807
24.1 The LLM Lifecycle	807
24.2 Pre-Training: Learning to Generate Text One Word at a Time .	808
24.3 Building Intuition: Character-Level Text Generation	811
24.4 The Scale Approach: How Bigger Became Better	813
24.5 Choosing the Right Model for Your Application	814
24.6 Distillation, Fine-tuning and Quantization	817

<i>Contents</i>	17
24.6.1 Quantization	817
24.6.2 Fine-tuning	818
24.6.3 Model Distillation: Knowledge Transfer	819
24.6.4 The Rise of Small Language Models	824
24.7 Evaluating Model Performance	825
24.8 Post-training Techniques	827
24.8.1 Chain-of-Thought and Chain of Reasoning	831
24.8.2 Reflexion	832
24.8.3 Non-Linear Reasoning Capabilities	832
24.9 Alignment	834
24.9.1 Reinforcement Learning from Human Feedback (RLHF) .	834
24.9.2 Direct Preference Optimization (DPO)	835
24.9.3 Constitutional AI	836
24.9.4 Safety Alignment	836
24.9.5 Reward Hacking and the Alignment Tax	837
24.10 Data quality and quantity	838
24.11 Dealing with Context Window Limitations: Context Engineering	840
24.11.1 The Lost-in-the-Middle Problem	841
24.11.2 When to Use Long Context vs. RAG vs. Summarization	841
24.11.3 Attention Efficiency: FlashAttention and Ring Attention	842
24.11.4 Retrieval-Augmented Generation	842
24.11.5 Advanced RAG Variants	844
24.11.6 Solving Lost-in-the-Middle	845
24.11.7 Caching and Compression	847
24.11.8 Neural Memory Systems	848
24.11.9 Evaluation and Deployment	849
24.12 Combining Techniques for Optimal Performance	850
24.12.1 Summary	852

25 AI Agents	855
25.1 LLM Agents	855
25.2 Agents with Personality	859
25.3 Agent Orchestration	859
25.3.1 Orchestration Patterns	861
25.3.2 Communication and State Management	862
25.4 AI Agent Training and Evaluation Methods	863
25.4.1 Categories of Agent Environments	864
25.4.2 Fundamental Evaluation Challenges	865
25.4.3 Evaluation Methodologies	865
25.4.4 Domain-Specific Benchmarks	866
25.4.5 The Human Role in Agent Evaluation	867
25.5 Agent Safety	868
25.5.1 Red-Teaming and Vulnerability Assessment	870
25.6 Robots	872
25.6.1 Challenges Unique to Embodied Agents	872
25.6.2 Modern Approaches to Robotic Intelligence	873
25.7 Conclusion	873
IV Appendices	875
26 Linear algebra and multivariate normal toolkit	877
26.1 Vectors, matrices, and dimensions	877
26.2 Transpose	877
26.3 Matrix-vector and matrix-matrix multiplication	877
26.4 Inner product and dot product	878
26.5 Norms	878
26.6 Inverse and matrix inversion	878
26.7 Determinant	878
26.8 Trace	879
26.9 Positive definiteness	879

<i>Contents</i>	19
26.10Eigenvalues and eigenvectors	879
26.11Singular value decomposition (SVD)	879
26.12A key quadratic form identity (Gaussian exponent)	880
References	881
References	881



Preface

Modern AI rests on three foundations: Bayesian reasoning, statistical learning, and deep neural networks. This book develops all three—starting from probability and decision theory, progressing through classical machine learning, and ending with transformers and autonomous agents. Whether you’re a business analyst seeking operational intuition or an engineer building systems, this book provides mathematical depth and practical fluency. Throughout, uncertainty is a first-class citizen: the foundation for better algorithms and clearer thinking about data, models, and decisions.

The material draws from courses we teach to MBAs at the University of Chicago Booth School of Business and engineers at George Mason University. These two audiences demand different emphases—business students want operational intuition and decision frameworks; engineers want mathematical rigor and implementation details—but each gains from the other—the same Bayesian methods that power marketing analytics and financial risk modeling also drive robotics and autonomous systems.

When John McCarthy coined “artificial intelligence” in 1956, the field meant expert systems, hand-coded rules and algorithms. The foundations trace further back: Claude Shannon’s information theory, John von Neumann’s game theory and decision science, and Richard Bellman’s dynamic programming. The shift to data-driven learning—models that generalize from examples defines modern AI. The contrast:

Old AI	New AI
<p><i>IF raining THEN take_umbrella</i></p> <p>A fixed rule. It works only one way. Seeing an umbrella tells us nothing about rain.</p>	<p><i>Probability of umbrella, given rain</i></p> <p>Learned from data. It works backwards as well. Seeing an umbrella tells us it’s likely raining.</p>

For example, the Kalman filter navigated Apollo 11 to the moon in 1969. Today’s autonomous systems—from rocket landings to self-driving cars—rely on optimization and learning algorithms that would have been computationally infeasible then.

How we interact with AI as consumers has evolved through four stages:

1. *Search.* Early search engines answered a single question with a ranked list of webpages. The PageRank algorithm, developed by Google founders, used power iterations to rank these pages by relevance. Statistical tools like Kendall's tau and Spearman's rank correlation measured the similarity between the ranking and actual relevance.
2. *Suggestions.* The first popular suggestion algorithm was developed by Netflix. It used collaborative filtering to recommend movies to users based on their viewing history and that of others, easing the burden of choice.
3. *Summaries.* Systems like ChatGPT go beyond retrieval: they synthesize and generalize across domains.
4. *Agents.* Autonomous systems that perceive their environment, reason about goals, and take actions—orchestrating tools to execute multi-step tasks without step-by-step human guidance.

Building systems at each stage requires the same foundations: probabilistic reasoning to handle uncertainty, statistical learning to extract patterns, and scalable computation to train models. This book develops all three.

The book is organized into three parts:

- *Part 1: Bayesian Learning:* Probability, decision theory, and Bayesian inference.
- *Part 2: Statistical Learning:* Pattern-matching algorithms including regression, decision trees, and generalized linear models.
- *Part 3: Deep Learning:* Neural network architectures, optimization via gradient descent, convolutional networks for vision, natural language processing, large language models, and autonomous agents.

Chapters include derivations, Python/R code, and problems and case studies.

The Modern AI Playbook

If you tell me precisely what it is a machine cannot do, then I can always make a machine which will do just that. —John von Neumann (1956)

When you open an Amazon page, there are many personal suggestions of goods to purchase. By analyzing previous product pages visited and purchases made by you and others who have bought similar products, Amazon uses AI and machine learning to predict what might interest you the next time you shop. When you apply for a loan online, you typically get an immediate answer after filling out an application. The information you provide, combined with your credit history pulled from a credit bureau, is used by a predictive model to determine with high confidence whether you are likely to default on the loan.

What do Amazon and the finance industry have in common? They all use AI-driven methods to improve operations. Automating routine workflows enables organizations to scale operations and tackle challenges that remain too complex for human labor alone. AI-driven methods are becoming increasingly important in all industries.

Because these technologies now drive core business value, organizations are competing aggressively for the limited pool of experts capable of building them. The demand for AI talent translates into substantial compensation premiums. According to the 2025 [AI Talent Salary Report](#), AI professionals command a median salary of \$160,000 annually (28% premium over traditional tech roles). Specialization matters: LLM engineers earn 25-40% more than general ML engineers, while AI Safety specialists have seen 45% salary increases since 2023.

What are the key ingredients of AI-driven methods? At the heart of modern AI lie three fundamental pillars:

- *Bayesian Learning:* A framework for reasoning under uncertainty. Bayesian methods quantify what we know and don't know, telling us how to update beliefs as new evidence arrives. This is essential in applications from drug discovery to climate modeling, where uncertainty is as important as the prediction itself.
- *Statistical Learning:* The collection of algorithms—from regression to decision trees—that extract patterns from data. These tools provide the mathematical rigor for recognizing relationships and making predictions.

- *Deep Learning*: Neural network architectures that learn hierarchical representations of data. By composing simple operations across many layers, these models capture complex, non-linear relationships in high-dimensional data, powering capabilities like vision, language understanding, and autonomous agents.

These software foundations are powered by *High-Performance Computing Infrastructure*. GPUs, originally designed for gaming, now make it feasible to process massive datasets and train billion-parameter models.

The interplay creates a virtuous cycle: computing power enables larger models (Deep Learning), while mathematical frameworks (Bayesian and Statistical Learning) ensure these models reason correctly and generalize well. Let's clarify the terminology used to describe this landscape.

- *Artificial Intelligence (AI)*, coined by John McCarthy in 1955, is the broad science of creating intelligent machines. It encompasses everything from the logic-based “Old AI” to modern data-driven approaches.
- *Machine Learning (ML)* is the subset of AI that focuses on learning from data. Rather than using explicit rules, ML systems find statistical structures in examples to make predictions or decisions.
- *Deep Learning* is a specialized branch of ML using neural networks with many layers (“deep”) to learn abstract representations of data, such as features in an image or concepts in text.

Data Science is the interdisciplinary field that uses these tools—along with statistics and software engineering—to extract insights from data.

Over the last three decades, a major shift has occurred from “hardcoded” expertise to learned patterns. IBM’s Deep Blue (1997) played chess using rules and heuristics crafted by grandmasters. In contrast, DeepMind’s AlphaGo Zero (2017) learned to play Go, Chess, and Shogi solely by playing against itself, deriving strategies superior to human knowledge without any human instruction. This transition—replacing engineered rules with learned rules—defines the modern AI era.

The key concept behind many modern AI systems is pattern-recognition. A “pattern” is a prediction rule that maps an input to an expected output, and “learning a pattern” means fitting a good prediction rule to a data set. In AI, prediction rules are often referred to as “models.” The process of using data to find a good prediction rule is often called “training the model.” Mathematically, we can express this as learning a function f that maps inputs x to outputs y , so that $y = f(x)$.



Figure 1: Pattern Recognition Process

For instance, in a large language model, x represents a question and y represents the answer. In a chess game, x represents the board position and y represents the best move. The learning process involves finding the function f that best captures the relationship between inputs and outputs by examining many examples of input-output pairs in a training dataset. Deep learning excels at discovering complex, nonlinear functions f when the relationship between x and y is too intricate to specify manually—such as the mapping from raw pixel values to semantic image content, or from question text to answer text.

Generative AI

By 2025, Generative AI has moved beyond hype into practical application across personal and professional domains. A Harvard Business Review article by Marc Zao-Sanders, “How People Are Really Using Gen AI in 2025,” reveals a notable trend: the top use cases have shifted from purely technical applications toward emotive and personal uses. “Therapy/companionship” now leads the list, followed by “Organizing my life,” “Finding purpose,” “Enhanced learning,” and “Generating code (for pros).”

Users are leveraging Gen AI for mental health support (especially in regions with limited access to therapists), daily habit planning, study guides, meal planning, travel itineraries, and drafting appeal letters. Gen AI users are also developing a deeper understanding of the technology’s limitations, including concerns around data privacy and over-reliance.

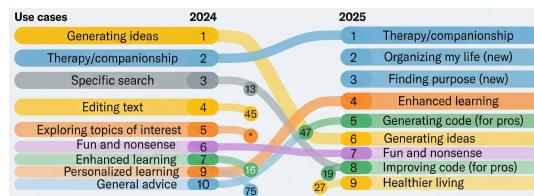


Figure 2: Top 10 Gen AI Use Cases in 2025. Source: Harvard Business Review, “How People Are Really Using Gen AI in 2025”, April 9, 2025.

The computer therapist is not something new. In 1966, Joseph Weizenbaum created **ELIZA**, a computer program that could simulate a conversation with a psychotherapist. ELIZA used simple pattern matching to respond to user inputs, creating the illusion of understanding. The program worked by identifying keywords in user statements and transforming them into questions or reflective responses. For example, if a user typed “I am sad,” ELIZA might respond with “Why do you think you are sad?” or “Tell me more about being sad.” While it was a groundbreaking achievement at the time, it lacked true comprehension and relied on scripted responses.

What surprised Weizenbaum was not just that ELIZA worked, but how readily people attributed human-like understanding to the program. Users began forming emotional attachments to ELIZA, sharing deeply personal information and believing the computer genuinely cared about their problems. Some even requested private sessions without Weizenbaum present. This phenomenon, now known as the *ELIZA effect*, describes the human tendency to unconsciously assume computer behaviors are analogous to human behaviors, even when we know better intellectually.

The ELIZA effect reveals something profound about human psychology: we are predisposed to anthropomorphize systems that exhibit even rudimentary conversational abilities. This has significant implications for modern AI systems. Today’s large language models like ChatGPT and Claude are vastly more sophisticated than ELIZA, yet they still operate through pattern matching and statistical prediction rather than genuine understanding. However, their responses are so fluent and contextually appropriate that the ELIZA effect is amplified dramatically. Users often attribute consciousness, emotions, and intentionality to these systems, leading to both beneficial therapeutic interactions and concerning over-reliance on AI for emotional support.

Understanding the ELIZA effect is crucial as we navigate the current AI landscape. While AI can provide valuable assistance for mental health support, learning, and personal organization, we must remain aware that these systems are sophisticated pattern matchers rather than conscious entities. The therapeutic value may be real—many users do find comfort and insight through AI interactions—but it stems from the human capacity for self-reflection prompted by the conversation, not from genuine empathy or understanding on the machine’s part.

In his talk on “Why are LLMs not Better at Finding Proofs?”, Timothy Gowers observes that LLMs can narrow down search spaces but falter when stuck, relying on intelligent guesswork rather than systematic problem-solving. Unlike humans, who respond to failed attempts with targeted adjustments based on what went wrong, LLMs often make another guess uninformed by previous failures. Humans build solutions incrementally; LLMs tend to skip intermediate steps and jump directly to answers, missing the structured, iterative reasoning that characterizes human problem-solving.

Despite these limitations in systematic reasoning, deep learning excels through pattern recognition. Consider the classical three-body problem in physics, which has resisted analytical solution for centuries. Deep neural networks have approximated solutions by learning the underlying dynamics from training data, generating accurate orbital trajectory predictions where closed-form solutions remain elusive.

While Deep Learning excels at recognizing patterns, we need a robust framework for reasoning about the uncertainty inherent in these predictions. This brings us to the Bayesian perspective.

Levels of Automation: AIQ, AGI, ASI

We can categorize AI systems by their level of autonomy. **AIQ** (Artificial Intelligence Quotient) represents the current paradigm where humans and machines work synergistically. As described by Nicholas G. Polson and Scott (2018), this intelligence augmentation allows humans to offload routine tasks while retaining strategic control. Current systems, including LLMs, fall into this category—they are powerful tools that amplify human capability but lack independent volition.

In contrast, **AGI** (Artificial General Intelligence) refers to hypothetical systems matching human performance across all cognitive tasks, while **ASI** (Artificial Superintelligence) describes systems surpassing human intelligence in every dimension. While AGI and ASI remain topics of intense research and debate, this book focuses on the practical, mathematical foundations of the AIQ systems building the modern economy today.

Software Development: The First Frontier for Agents

Andrej Karpathy’s talk, “[Software Is Changing \(Again\)](#),” illustrates a clear “product-market fit” for AI agents. Software development has become the first domain where agents have demonstrated significant, quantifiable impact, fundamentally transforming how code is written. Karpathy describes this new era as “Software 3.0,” where natural language becomes the primary interface.

Paradigm	“Program” is...	Developer’s main job	Canonical depot
<i>Software 1.0</i>	Hand-written code	Write logic	GitHub

Paradigm	“Program” is...	Developer’s main job	Canonical depot
<i>Software 2.0</i>	Neural-net <i>weights</i>	Curate data & train	Hugging Face
<i>Software 3.0</i>	Natural-language <i>prompts</i>	Guide & Verify	Prompt libraries

This success in software engineering serves as a blueprint. The patterns established here—iterative planning, tool use, and verification—are already spreading to high-stakes fields like **finance** and **healthcare**, where agents will likely follow a similar trajectory of adoption.

AI Agents: From Prediction to Action

The concept of software acting on our behalf has deep roots. Hal Varian’s 2010 framework of “Computer Mediated Transactions” (Varian 2010) foresaw a world where computers would automate not just calculation, but routine decision-making processes—monitoring markets, managing inventory, and co-ordinating complex transactions. This vision is now being realized and surpassed by modern *AI Agents*.

While a Language Model (LLM) is like a brain in a jar—capable of thought (prediction) but isolated—an *Agent* is an LLM equipped with tools and agency. Agents can browse the web, execute code, query databases, and interact with other software systems to complete multi-step workflows. Unlike traditional rigid software, agents dynamically adapt their behavior: if a first attempt fails, they can “reason” about the error and try a different strategy.

Building reliable agents requires robust *orchestration frameworks* that manage memory, planning, and tool execution. An example of a company that focuses on AI workloads is Nebius. Unlike traditional cloud architecture designed for general web services, their platform rethinks how data is stored, processed, and computed upon to support the massive parallel throughput required by modern AI models. This shift—from static models to dynamic agents—marks the transition from software that answers questions to software that iteratively plans and executes tasks in the digital and physical economy.

Physical AI: Embodied Intelligence

While Large Language Models have mastered the digital realm of text and code, a new frontier is emerging: Physical AI. This field aims to give artificial intelligence a physical body, enabling it to perceive, understand, and interact with the tangible world. Unlike digital AI that processes symbols, Physical AI

must contend with the chaotic laws of physics—gravity, friction, and unstructured environments. This represents a significant leap from “narrow” robotics, which followed rigid, pre-programmed instructions, to “embodied” intelligence that can learn, adapt, and operate alongside humans.

The convergence of vision-language models and advanced robotics is accelerating this shift. Companies like Tesla with Optimus, Boston Dynamics, and Figure are developing humanoid robots capable of performing complex tasks, from folding laundry to assembling cars. A key enabler is “sim-to-real” reinforcement learning, where robots train for millions of hours in hyper-realistic physics simulations (like NVIDIA’s Isaac Lab) before downloading those learned skills into a physical body. This allows them to master movements that would take decades to learn in the real world.

The Human Dimension: Dignity and Economics

“I visualize a time when we will be to robots what dogs are to humans. And I am rooting for the machines.” - Claude Shannon

As we delegate more authority to algorithms, the warnings of the cybernetics era become urgently relevant. *Norbert Wiener*, the mathematician who founded cybernetics and whose work laid foundations for modern control systems and artificial intelligence, was among the first to recognize both the power and peril of automation. Writing in *The Human Use of Human Beings* (1950), Wiener articulated a prescient warning:

If we combine our machine potentials of a factory with the valuation of human beings on which our present factory system is based, we are in for an Industrial Revolution of unmitigated cruelty. We must be willing to deal in facts rather than fashionable ideologies if we wish to get through this period unharmed.

Wiener’s concern was fundamentally about human identity and dignity. If workers derive their sense of worth from their role as factory laborers, and automation eliminates those roles, what becomes of their identity? This was not merely an economic question about displaced workers finding new employment—it was a deeper psychological and existential challenge. Wiener recognized that the transition to an automated economy would require not just retraining programs, but a fundamental reimaging of how humans find meaning and value in a world where machines perform an ever-expanding range of tasks.

Crucially, Wiener rejected the notion that automation would deliver humanity into a comfortable retirement:

The world of the future will be an even more demanding struggle against the limitations of our intelligence, not a comfortable hammock in which we can lie down to be waited upon by our robot slaves.

This vision stands in stark contrast to utopian fantasies of automated abundance. Wiener foresaw that as machines took over routine cognitive and physical tasks, the remaining challenges would become more abstract, more complex, and more demanding of human creativity and judgment. The age of AI would not eliminate work—it would transform it into work that pushes against the very boundaries of human capability.

John Maynard Keynes, writing during the Great Depression in his 1930 essay *Economic Possibilities for our Grandchildren* (1930), offered a remarkably optimistic counterpoint. Keynes predicted that within a century—roughly by 2030—technological progress and capital accumulation would “solve the economic problem” for humanity. By this he meant that productivity gains would become so substantial that meeting humanity’s basic material needs would require only minimal labor. Keynes envisioned a future where people might work perhaps fifteen hours per week, devoting the remainder of their time to leisure, culture, and the pursuit of fulfilling activities.

Keynes distinguished between *absolute needs*—those we feel regardless of others’ circumstances, such as food, shelter, and safety—and *relative needs*—our desire to feel superior to our fellows. He argued that while relative needs are insatiable, absolute needs could be satisfied through technological abundance. Once this occurred, humanity would face a new challenge: learning to live wisely with leisure. Keynes worried that without the structure and purpose provided by work, many people would struggle to find meaning. He wrote that humanity would need to cultivate the “art of life itself” and learn to value activities pursued for their own sake rather than for economic gain.

Keynes’s prediction that technology would dramatically increase productivity proved remarkably accurate. However, his assumption that increased productivity would translate into reduced working hours has not materialized as he expected. Rather than collectively choosing leisure, advanced economies have channeled productivity gains into increased consumption, higher living standards, and the expansion of service industries. The phenomenon of “Veblenian” conspicuous consumption helps explain why.

Thorstein Veblen, writing even earlier in *The Theory of the Leisure Class* (1899), offered a more cynical analysis of how elites use both leisure and consumption to signal status. Veblen introduced the concept of *conspicuous consumption*—the purchase of goods and services primarily to display wealth and social status rather than to satisfy genuine needs. The leisure class, in Veblen’s analysis, derives its social standing not from productive labor but from the ostentatious display of time and resources devoted to non-productive activities.

Veblen's insight reveals why Keynes's vision of universal leisure has not materialized. In modern economies, work serves not only to produce income for consumption but also to confer identity, status, and social belonging. High-status professionals often work longer hours than necessary for material sustenance precisely because their work signals competence, dedication, and membership in elite circles. The "leisure" time that technology has created has often been filled not with Keynesian cultivation of the art of life, but with Veblenian status competitions—from luxury travel photographed for social media to the accumulation of credentials through continuous education.

Moreover, as AI automates routine tasks, the remaining human work increasingly involves activities that are themselves forms of status display: strategic decision-making, creative innovation, and high-stakes problem-solving. These activities signal membership in cognitive elites in ways that parallel Veblen's leisure class. The AI era has not eliminated status competition through work—it has transformed the nature of the work that confers status.

In his later work, *The Engineers and the Price System* (1921), Veblen examined the role of technical experts in modern industrial society. He distinguished between two fundamentally different modes of economic organization: the industrial system, driven by engineers and technical specialists focused on efficient production, and the price system, controlled by financiers and business owners focused on profit extraction. Veblen's analysis is remarkably prescient for understanding tensions in the contemporary AI economy. Today's "engineers"—the data scientists, machine learning researchers, and software developers building AI systems—possess technical knowledge that enables unprecedented productive capabilities. Yet the deployment of these capabilities is mediated through corporate structures optimized for profit maximization rather than social welfare.

The synthesis of these perspectives suggests that successfully navigating the AI transition requires more than technical solutions or economic policies. It requires cultivating new sources of meaning, identity, and social connection that are not solely dependent on traditional employment. It requires resisting purely Veblenian status competitions in favor of Keynesian cultivation of intrinsically valuable activities. And it requires heeding Wiener's warning that the future will demand more, not less, of our intelligence, creativity, and ethical judgment—even as machines handle an expanding range of routine tasks.

Large Language Models (LLMs)

The most visible manifestation of the new AI age is Large Language Models. ChatGPT reached 100 million users in 2 months after its November 2022 launch—a milestone that took the internet 7 years and television 13 years. Unlike previous innovations requiring infrastructure changes, AI chatbots provide immediate value through simple web interfaces.

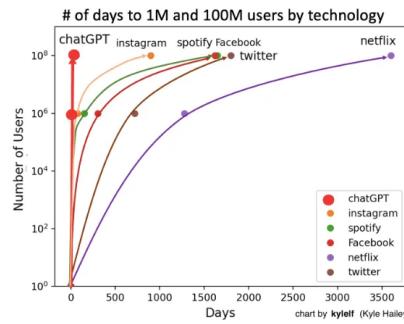


Figure 3: Source: <https://johnnosta.medium.com/the-most-important-chart-in-100-years-1095915e1605>

The algorithmic foundations of deep learning have existed for decades: Kolmogorov’s superposition theorem (1956), Robbins-Monro’s stochastic approximation (1951), Tikhonov regularization (1940s), Polyak momentum optimization (1964), and Galushkin’s backpropagation (1972) constitute the “old math” that enables modern AI alongside new GPU chips. The 2012 breakthrough with AlexNet would have been impossible without GPUs performing thousands of matrix multiplications simultaneously and convolutional neural networks developed by Fukushima in the early 80s. Current models rely on training clusters of thousands of interconnected GPUs working for weeks.

GPU computational power has grown over 400,000x in two decades—from the 0.23 TeraFLOPS of 2006’s GeForce 7900 GTX to projected 100 PetaFLOPS for 2027’s Nvidia Rubin Ultra. This exponential growth, driven successively by gaming, cryptocurrency mining, and AI video generation, has made GPUs the engines of the deep learning era. Modern architectures feature specialized tensor cores and mixed-precision arithmetic (FP8/FP4) optimized for AI workloads rather than traditional floating-point operations.

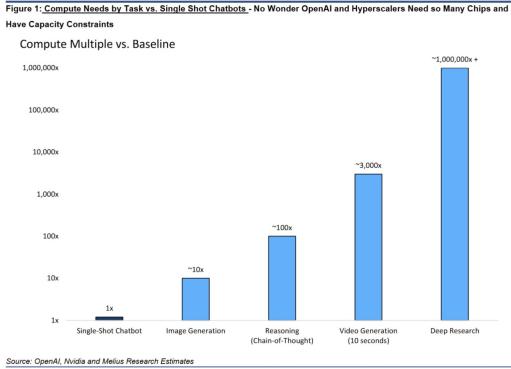


Figure 4: Compute requirements scale exponentially across AI tasks. Source: Michael Dell

The computational demands of AI tasks scale exponentially (Figure 4): while a single-shot chatbot represents the baseline (1x), image generation requires $\sim 10x$ more compute, reasoning tasks need $\sim 100x$, video generation demands $\sim 3,000x$, and deep research capabilities require over 1,000,000x. Commercial LLMs measure computation in tokens—units of text processed as inputs or outputs—rather than traditional FLOPS. Early LLMs required fine-tuning for specific tasks; current models work effectively through in-context learning, where task-specific instructions and data are provided as context.

This exponential scaling illustrates a modern instance of *Jevons paradox*. In 1865, economist William Stanley Jevons observed that improvements in steam engine efficiency did not reduce coal consumption—instead, efficiency gains made coal-powered machinery economically viable for more applications, increasing total coal use. The same dynamic applies to AI compute: as models become more efficient and inference costs drop, usage expands faster than efficiency improves. Cheaper tokens enable longer conversations, more complex reasoning chains, and new applications like real-time video generation that were previously cost-prohibitive. The result is that total compute consumption continues to grow despite—and because of—efficiency improvements.

While GPUs have driven the current AI innovation, the quest for *quantum supremacy* represents the next frontier. As Feynman’s seminal paper first discussed, the principles of quantum physics suggest a new foundation for computation. The work by Nick Polson, Sokolov, and Xu (2023) on quantum Bayesian computation explores quantum algorithms that promise exponential speed-ups for Bayesian and neural network methods. While stable quantum computers remain in development, the algorithms are already being designed.

Business Models and Market Impact

Four distinct business models have emerged among leading LLM providers:

- *Anthropic* (Claude) focuses on enterprise safety, using Constitutional AI training methods that emphasize nuanced reasoning and acknowledgment of uncertainty—attracting regulated industries and research institutions.
- *Google* (Gemini) leverages deep integration with its ecosystem—Search, Workspace, Android, and Cloud—to distribute AI capabilities at scale. With access to proprietary data (YouTube, Maps, Scholar) and custom TPU hardware, Google competes on both consumer reach and enterprise infrastructure.
- *OpenAI* (ChatGPT) pioneered the subscription-plus-API model, combining consumer subscriptions (\$20/month for Plus) with enterprise API pricing. Microsoft's strategic partnership provides exclusive cloud infrastructure.
- *Perplexity* reimagines search as a citation-based conversational engine, synthesizing information from multiple sources with attribution rather than returning links.

LLMs have found immediate applications across industries: customer service teams report 30-50% cost reductions with chatbots handling routine inquiries; coding assistants help developers complete tasks 55% faster; legal firms compress weeks of contract review into hours. The pattern is consistent: LLMs excel at processing, summarizing, and generating text-heavy work that previously required skilled human labor. Most deployments involve human oversight, but they fundamentally change the economics of knowledge work.

Bayes: Evidence as Negative Log-Probability

Imagine you're searching for something lost—a missing ship, a hidden treasure, or a city abandoned centuries ago. You have multiple clues: historical documents, geological surveys, satellite imagery, and expert opinions. How do you combine all these disparate pieces of evidence into a coherent search strategy? This is exactly the type of problem where Bayesian reasoning shines, and it's a powerful framework that underlies many modern AI applications.

The Bayesian approach provides a principled mathematical framework for updating our beliefs as new evidence arrives. At its core is Bayes' rule, which tells us how to revise the probability of a hypothesis given new data:

$$P(\text{hypothesis} \mid \text{data}) = \frac{P(\text{data} \mid \text{hypothesis}) \times P(\text{hypothesis})}{P(\text{data})}$$

While this formula is elegant, what makes Bayesian reasoning especially powerful is a simple mathematical trick: when we work with *logarithms* of probabilities, combining evidence becomes as simple as addition. Taking the logarithm of both sides of Bayes' rule gives us:

$$\log P(\text{hypothesis} \mid \text{data}) = \log P(\text{data} \mid \text{hypothesis}) + \log P(\text{hypothesis}) - \log P(\text{data})$$

This transformation reveals that the log-posterior (our updated belief) is simply the sum of the log-likelihood (evidence from data) and the log-prior (our initial belief), minus a normalization constant. In other words, on the log scale, we're just adding up different sources of evidence. Each piece of information contributes its "weight" to the total, and we combine them linearly.

This additive property has profound practical implications. When you have multiple independent sources of evidence—say, historical documents, geological surveys, and geophysical measurements—each contributes a term to the sum. Strong evidence adds a large positive contribution, weak evidence adds little, and contradictory evidence subtracts from the total. The beauty is that the mathematical framework handles all the bookkeeping automatically.

A remarkable application of this principle comes from the world of mineral exploration. In 2022, [Aurania Resources announced](#) that they had found the location of Logroño de los Caballeros, a "lost city" of Spanish gold miners that had been abandoned in the jungles of Ecuador for over 400 years. The discovery was made possible by Bayesian search theory, developed by Larry Stone who has a remarkable track record of finding lost objects—including the USS Scorpion nuclear submarine and Air France Flight 447.

Larry Stone's approach to finding Logroño exemplifies how Bayesian reasoning combines multiple sources of evidence. The team assembled a mountain of heterogeneous information:

- *Historical documents*: Spanish colonial records from the 1580s-1590s describing Logroño's location relative to rivers and other settlements
- *Archaeological evidence*: A 1574 map by Mendez showing approximate locations
- *Geological data*: Stream sediment samples analyzed for gold content
- *Geophysical surveys*: Magnetic and radiometric measurements
- *Modern geography*: LiDAR topographic data and current river systems
- *Geochemical patterns*: Distribution of minerals indicating potential gold sources

Each of these information sources provided a "clue" that was more or less reliable, more or less precise, and potentially contradictory with others. How

do you reconcile a 450-year-old account that “Logroño was half a league from the Rio Zamora” with geological evidence suggesting gold-bearing formations in a different area?

Bayesian search theory provides the answer. Bayesian reasoning assigns each piece of evidence a reliability weight and used Bayes’ rule to generate probability maps. Historical documents considered highly reliable (such as official Spanish reports) contributed strongly to the probability distribution, while more ambiguous sources contributed less. Larry Stone, explained: “Our success in integrating historical documents with scientific data using Bayesian methods opens a range of potential applications in the mineral and energy exploration sectors.”

The power of this approach became clear when they combined evidence that initially seemed contradictory. A critical breakthrough came from multiple corroborating accounts: Juan Lopez de Avendaño reported in 1588 that Logroño was half a league from the Rio Zamora; that same year, two soldiers drowned crossing “the river” to fight an uprising; in the mid-1590s, seven soldiers drowned trying to reach a downstream garrison; and a 1684 Jesuit account described an elderly woman who remembered hearing Logroño’s church bells from her village at the mouth of the Rio Zamora. Each piece of evidence individually was ambiguous—which river? how far is “half a league”?—but together they pointed to a specific location along the Rio Santiago valley.

On the log-probability scale, each piece of evidence either added to or subtracted from the likelihood of different locations. Strong, consistent evidence (multiple drowning accounts suggesting a major river crossing) added significant weight. Weak or contradictory evidence contributed less. The final probability map was literally the sum of these contributions, with the peak probability occurring where the most evidence converged. Figure 5 shows the likelihood ratio surfaces generated for copper, silver, and gold deposits—visual representations of how different evidence sources combine to create probability distributions across the search area.

Combined Likelihood Ratio Maps

To combine the evidence, we multiply the Likelihood Ratios: $LRLitho \times LRMag \times LRGeochem$

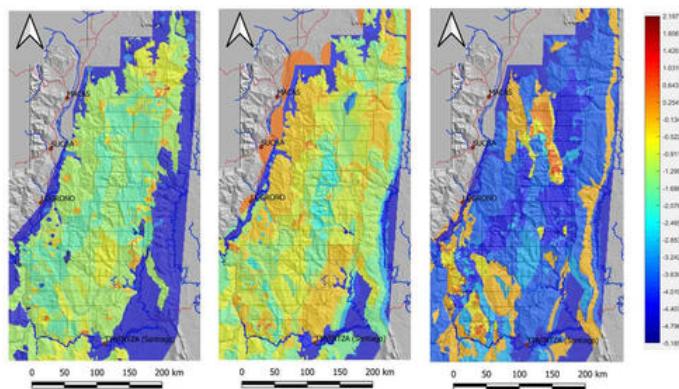


Figure 5: Likelihood ratio surfaces generated by Metron showing potential locations for copper, silver, and gold deposits in Aurania's concession area. These heat maps visualize how Bayesian analysis combines multiple sources of geological and geophysical evidence into a single probability distribution. Warmer colors indicate higher likelihood ratios where multiple pieces of evidence converge. Source: Metron Inc., via [MIT Sloan](#)

The result was dramatic: Bayesian reasoning generated probability maps that identified the Rio Santiago valley as the most likely location of Logroño, and subsequent fieldwork confirmed extensive alluvial gold deposits and active artisanal mining exactly where the Bayesian analysis predicted. As Dr. Keith Barron, Aurania's CEO, noted: "This key discovery can ultimately lead us to Logroño's gold source." The location that seemed to reconcile all the disparate evidence—Spanish colonial records, drowning accounts, geological surveys, and modern geography—turned out to be correct.

This example illustrates why the Bayesian framework is so powerful in modern AI applications. Machine learning models constantly face the challenge of combining multiple sources of information: pixels in different regions of an image, words in different parts of a sentence, measurements from different sensors. The additive property of log-probabilities provides an efficient computational framework for this fusion. When you train a deep learning model, the loss function essentially measures how well the model combines evidence from the training data with prior knowledge (encoded in the model architecture and regularization). Optimization algorithms adjust model parameters to maximize this combined evidence, updating beliefs exactly as Bayes' rule prescribes.

The mathematical elegance of working with log-probabilities extends beyond search problems. In natural language processing, transformer models compute attention weights that determine how much “evidence” each word provides about the meaning of other words. In computer vision, convolutional networks combine evidence from different receptive fields. In recommendation systems, collaborative filtering combines evidence from multiple users’ preferences. All of these applications benefit from the additive structure that log-probabilities provide.

From Beliefs to Decisions: Expected Utility

Bayesian reasoning is not merely about updating beliefs—it is fundamentally about making decisions under uncertainty. The Logroño example illustrates this: the ultimate question was not “what is the probability that gold lies at location X?” but rather “where should we drill?” This is a decision, not an estimation problem.

The expected utility framework bridges beliefs and actions. The idea is simple: for each possible action, we weight its value in each scenario by how likely that scenario is, then choose the action with the highest weighted average. In the Logroño case, drilling at a location where multiple evidence sources converge has higher expected value than drilling where only one source points—even if that single source seems compelling.

This framework unifies diverse applications: A/B testing asks “which variant should we deploy?” given uncertain conversion rates. Portfolio optimization asks “how should we allocate capital?” given uncertain returns. Medical treatment selection asks “which therapy should we prescribe?” given uncertain patient response. In each case, the decision-maker must specify preferences—the relative costs of false positives versus false negatives, the risk tolerance for financial losses, or the tradeoff between treatment efficacy and side effects.

Decision theory is integral to the Bayesian approach precisely because probability alone is insufficient for action. Two analysts may agree on the posterior probability of a drug’s effectiveness yet disagree on whether to approve it, depending on how they weigh the costs of denying a beneficial treatment against approving a harmful one. The Bayesian framework makes these tradeoffs explicit and principled.

Statistical Learning: Patterns in Data

While Bayesian methods provide the framework for reasoning under uncertainty, statistical learning supplies the algorithmic toolkit for extracting patterns from data. The core problem is simple to state: given examples of inputs x and outputs y , find a function f such that $y \approx f(x)$ for new, unseen inputs.

Regression is the workhorse of prediction. Linear regression models the relationship as $y = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p + \epsilon$, where the coefficients β quantify how each input variable influences the output. Despite its simplicity, linear regression remains the starting point for most predictive analyses—from housing price prediction to demand forecasting—because its interpretability allows domain experts to understand and trust the model.

Tree-based methods offer an alternative that naturally captures non-linear relationships and interactions. A decision tree recursively partitions the input space, asking questions like “is income > \$50K?” and “is age < 30?” to create regions with homogeneous outcomes. Random forests average many such trees trained on bootstrap samples, reducing variance while maintaining flexibility. Gradient boosting sequentially builds trees that correct the errors of previous ones. These ensemble methods dominate Kaggle competitions and production ML systems because they require minimal tuning, handle mixed data types gracefully, and provide variable importance measures that aid interpretation.

The tension between model complexity and generalization is formalized by the *bias-variance tradeoff*. Simple models (high bias) may miss important patterns; complex models (low variance in training but high variance in prediction) may overfit to noise. Cross-validation—holding out data for testing—provides a principled approach to model selection: we choose the model complexity that minimizes prediction error on data the model has not seen during training.

These classical statistical learning methods form the foundation for understanding deep learning. Neural networks can be viewed as highly flexible function approximators that automatically learn useful representations of x . The same principles—minimizing prediction error, regularizing to prevent overfitting, validating on held-out data—apply across the spectrum from linear regression to billion-parameter language models.

Examples: AI in Action

The following examples demonstrate how deep learning extracts meaningful patterns from complex data—creative synthesis, sports analytics, high-speed sailing optimization, and scientific discovery.

Example 0.1 (The Next Rembrandt). In 2016, a “new” Rembrandt painting was unveiled in Amsterdam—not discovered in an attic, but generated by algorithms. The project was the brainchild of Bas Korsten, creative director at J. Walter Thompson Amsterdam.

The portrait emerged from 18 months of analysis of 346 paintings and 150 gigabytes of digitally rendered graphics. Everything about the painting—from the subject matter (a Caucasian man between 30 and 40) to his clothes (black, wide-brimmed hat, black shirt and white collar), facial hair (small mustache and goatee), and the way his face is positioned (facing right)—was distilled from Rembrandt’s body of work.

“A computer learned, with artificial intelligence, how to re-create a new Rembrandt right eye,” Korsten explains. “And we did that for all facial features, and after that, we assembled those facial features using the geometrical dimensions that Rembrandt used to use in his own work.”

A deep learning model learned the statistical distribution of “Rembrandtness,” capturing the artist’s style not as a set of rules, but as a probability distribution over pixel arrangements. The resulting image is a hallucination that fits this distribution perfectly—demonstrating Generative AI’s ability to synthesize high-fidelity creative artifacts.



Figure 6: Can you guess which image was generated by the algorithm?

Example 0.2 (Learning Person Trajectory Representations for Team Activity Analysis). Activity analysis in which multiple people interact across a large space is challenging due to the interplay of individual actions and collective group dynamics. A recently proposed end-to-end approach (Mehrasha et al. 2017) allows for learning person trajectory representations for group activity

analysis. The learned representations encode rich spatio-temporal dependencies and capture useful motion patterns for recognizing individual events, as well as characteristic group dynamics that can be used to identify groups from their trajectories alone. Deep learning was applied in the context of team sports, using the sets of events (e.g. pass, shot) and groups of people (teams). Analysis of events and team formations using NHL hockey and NBA basketball datasets demonstrate the generality of applicability of DL to sports analytics.

When activities involve multiple people distributed in space, the relative trajectory patterns of different people can provide valuable cues for activity analysis. We learn rich trajectory representations that encode useful information for recognizing individual events as well as overall group dynamics in the context of team sports.

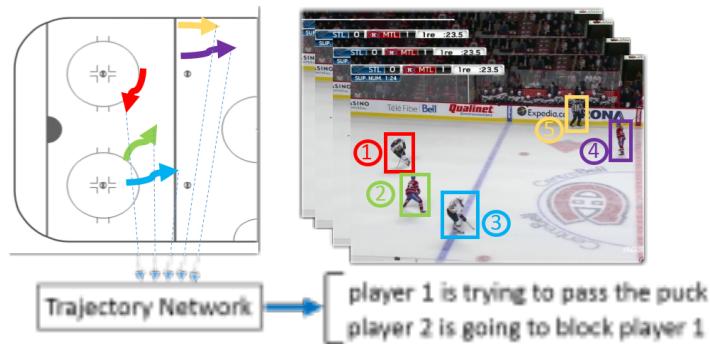


Figure 7: Hockey trajectory analysis visualization

Example 0.3 (EPL Liverpool Prediction). Liverpool FC has become a benchmark in football for integrating advanced data analytics into both their recruitment and on-field strategies. The Expected Possession Value (EPV) pitch map shown below displays the likelihood that possession from a given location will result in a goal, with red areas indicating high-value zones where Liverpool's chances of scoring increase significantly when they gain or retain the ball. Liverpool's analysts use these EPV maps to inform tactical decisions and player positioning, allowing the coaching staff to instruct players to press, pass, or move into these high-value zones.

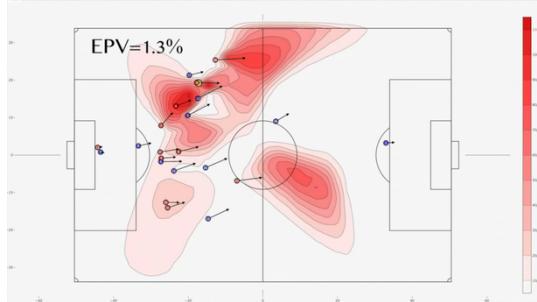


Figure 8: Expected Possession Value (EPV) pitch map

On April 26, 2019, Liverpool scored their fastest-ever Premier League goal—Naby Keita found the net just 15 seconds into the match against Huddersfield Town, setting the tone for a dominant 5-0 victory. This remarkable goal exemplified Liverpool’s data-driven approach to football. Keita’s immediate pressure on Huddersfield’s Jon Gorenc Stankovic was not random—it was a calculated move informed by analytics revealing Huddersfield’s vulnerability when building from the back under pressure. This demonstrates the effective application of Expected Possession Value (EPV) principles. Liverpool’s analysts systematically study opponent build-up patterns, using video and tracking data to predict where and how opponents are likely to play the ball from kick-off or in early possession phases. This intelligence allows Liverpool players to position themselves strategically for maximum disruption. When Huddersfield’s goalkeeper played out from the back, Keita was already moving to intercept, anticipating the pass route—a behavior that had been drilled through analytics-driven preparation and scenario planning.

Example 0.4 (SailGP: Analytics at 50 Knots). SailGP racing catamarans are floating data centers, instrumented with over 1,000 sensors generating 52 billion data points per race. Oracle co-founder Larry Ellison pioneered this fusion of sailing and analytics through his involvement with Oracle Team USA and the America’s Cup. His approach has shaped modern sailing competitions like SailGP.

The power of real-time analytics was dramatically demonstrated in the 2013 America’s Cup. Jimmy Spithill and ORACLE TEAM USA faced Emirates Team New Zealand, falling behind 8-1 in the best-of-17 series—a deficit no team had ever recovered from in America’s Cup history. The turning point came after the 8-1 loss when the team made a critical technological decision: they installed additional sensors throughout the boat to collect more comprehensive data about performance, wind conditions, and boat dynamics. These sensors provided real-time feedback that allowed precise adjustments to sailing strategy and boat configuration. With the enhanced data collection system in place, ORACLE TEAM USA won eight consecutive races to claim

the America’s Cup 9-8—one of the greatest comebacks in sporting history.

Modern SailGP boats build on this foundation. Key metrics like *Velocity Made Good (VMG)*—the speed towards the mark—are continuously recalculated based on wind shifts and currents. Tack and gybe optimization uses statistical modeling to determine optimal timing for direction changes. Layline calculations employ predictive analytics to minimize distance sailed. Pressure sensors combined with flow dynamics models calculate optimal hydrofoil position. Teams use Bayesian inference to update their models in real-time as new data arrives during races, creating a dynamic optimization system that continuously refines strategy.

The system performs “inverse inference” on the physical environment: observing the boat’s performance updates beliefs about unseen wind patterns, allowing crews to make optimal decisions in split seconds. Success depends as much on the ability to collect, process, and act on real-time data as on traditional sailing skills.



Figure 9: Emirates GBR SailGP Team

Example 0.5 (DeepMind’s Alpha Series: From Games to Science). DeepMind’s journey into reinforcement learning began with a focus on mastering games, not for the sake of play, but to test the limits of artificial intelligence. It started with *AlphaGo*, which stunned the world in 2016 by defeating 18-time world champion Lee Sedol at the ancient game of Go—a feat previously thought to be a decade away. The system’s creative “Move 37” demonstrated an alien form of intuition, suggesting that machines could transcend human imitation. This evolution continued with *AlphaZero*, which removed the need for human examples entirely. By playing millions of games against itself, it mastered Go, Chess, and Shogi from scratch in mere hours, discovering novel strategies that had eluded human grandmasters for centuries and proving that AI could learn to solve complex problems through pure trial and error.

The true power of these systems, however, lies in their ability to step outside the game board and tackle fundamental scientific challenges. *AlphaFold* ap-

plied the same principles to biology, solving the 50-year-old “protein folding problem” by predicting the 3D structures of nearly all known proteins, a breakthrough that is now accelerating drug discovery and our understanding of life itself. Most recently, *AlphaProof* has ventured into the realm of abstract reasoning, solving complex mathematical problems from the International Mathematical Olympiad. Together, these models demonstrate a profound shift: we are moving from AI that entertains us to AI that expands the frontiers of human knowledge, turning the intuition learned in games into tools for scientific discovery.

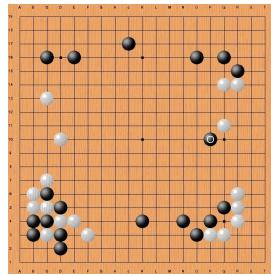


Figure 10: Alpha GO vs Lee Sedol: Move 37 by AlphaGo in Game Two

Anything as a Vector

Modern AI treats all information—text, images, audio, biological sequences—as *vectors*: lists of numbers that represent meaning in a high-dimensional space.

Tokenization and Embeddings

In Natural Language Processing (NLP), text is first broken into *tokens* (words or sub-words). Each token is then mapped to a vector. Unlike simple ID numbers (e.g., cat=1, dog=2), these learned vectors capture semantic relationships: the vector for “cat” is mathematically closer to “dog” than to “car.”

Example 0.6 (Semantic Relations). One of the most intriguing aspects of vector representations is their ability to capture semantic relationships through simple arithmetic operations. In natural language processing, this is famously illustrated by analogies such as “king - man + woman = queen,” where the difference between “king” and “man” encodes the concept of royalty, and adding “woman” shifts the meaning to “queen.” This property emerges because the learned vectors for words, phrases, or even entities are organized in

such a way that similar relationships are reflected as consistent directions in the high-dimensional space. The same principle applies beyond language: for example, in sports analytics, we might find that the vector for “Ovechkin” (a star hockey player) plus the vector for “Capitals” (his team) minus the vector for “Gretzky” (another legendary player) yields a vector close to “Oilers” (Gretzky’s team), capturing the underlying relationships between players and their teams.

$$\text{Ovechkin} + \text{Capitals} - \text{Gretzky} = \text{Oilers}$$

This ability to perform analogical reasoning with vectors is not limited to words or names—it extends to images, audio, and even structured data like chess positions. In computer vision, for instance, the difference between the vector representations of two images might correspond to a specific transformation, such as changing the background or adding an object. In recommendation systems, the vector difference between a user’s preferences and an item’s features can help identify the best match. These semantic relations, encoded as vector arithmetic, enable AI systems to generalize, reason, and make creative associations across domains. The power of this approach lies in its universality: once information is embedded in a vector space, the same mathematical tools can be used to uncover patterns and relationships, regardless of the original data type.

Vectorizing Chess

The vectorization concept becomes particularly clear when we examine how chess positions can be represented numerically. A chess board contains 64 squares, each of which can be empty or occupied by one of 12 different piece types (6 pieces \times 2 colors). We can represent any chess position as a vector by simply listing the contents of each square in order. For instance, we might use the encoding: empty=0, white pawn=1, white rook=2, ..., white king=6, black pawn=7, black rook=8, ..., black king=12. A chess position would then become a 64-dimensional vector like [8, 9, 10, 11, 12, 10, 9, 8, 7, 7, 7, 7, 7, 7, 7, 7, 7, 0, 0, 0, 0, 0, 0, 0, ...] representing the starting position with black pieces on the back rank, black pawns on the second rank, and so forth. More sophisticated representations might include additional dimensions for castling rights, en passant possibilities, or whose turn it is to move, creating vectors of 70 or more dimensions. This numerical representation allows chess engines to use the same mathematical operations that work for language or images. The AI can learn that certain vector patterns (piece configurations) are more advantageous than others, and it can mathematically compute how different moves transform one position vector into another. Modern chess engines like AlphaZero process millions of these position vectors to evaluate potential moves, demonstrating how any complex domain can be reduced to vector operations that computers excel at manipulating.

Modern engines like AlphaZero don't just "read" the board; they map these board-state vectors into a learned strategy space. Moves that lead to victory are clustered together, allowing the AI to "see" tactical patterns structurally, just as an LLM "sees" grammatical relationships.

Human vs. Machine Representations

While modern AI uses a universal vector language, human cognition relies on distinct, specialized systems. Informal experiments by Richard Feynman and John Tukey at Princeton in 1939 illustrate this contrast (Gleick 1992). They challenged each other to count time intervals while performing other tasks.

The results revealed a striking asymmetry: Feynman could read silently while counting but failed if he spoke; Tukey could recite poetry while counting but failed if he read. Unknown to them, they had discovered working memory's separate channels: Feynman counted "auditorily" (conflicting with speech), while Tukey counted "visually" (conflicting with reading).

This underscores a fundamental difference: biology evolved specialized channels (auditory vs. visual), whereas AI unifies all data—text, images, audio—into the same *vector operations*, allowing a single model to become truly multimodal.

How Machines Learn: The Loss Function

"Let us suppose we have set up a machine with certain initial instruction tables... One can imagine that after the machine had been operating for some time, the instructions would have altered out of all recognition... It would be like a pupil who had learnt much from his master, but had added much more by his own work." – Alan Turing (Turing 1950)

If vectors are the **language** of AI, then the **loss function** is its teacher. An AI model begins with random "instruction tables" (weights). It makes a prediction, compares it to the correct answer, and calculates a "loss"—a single number representing the error (e.g., the distance between the predicted pixel and the real pixel). Optimization algorithms (like Gradient Descent) then work backward, slightly adjusting the billions of weights to reduce this loss.

This process, repeated billions of times, essentially "compiles" data into software.

Example 0.7 (Example: Learning Language). Large Language Models (LLMs) master this through a simple objective: **predict the next token**. 1. **Input:** “The cat sat on the...” 2. **Prediction:** The model computes probabilities for every possible next word. 3. **Update:** If it predicts “car” (low probability for “mat”), the loss is high. The weights are adjusted to make “mat” more likely next time.

This simple mechanism—reducing prediction error—forces the model to internalize grammar, facts, and reasoning. To predict accurately, it must understand that “sat” implies an agent, “on” implies a location, and “mat” fits the context. As Turing predicted, the machine alters its own instructions to become a “pupil” that eventually surpasses its initial programming.

Why does this produce seemingly abstract knowledge? Because the cheapest way to predict language at scale is to internalize the latent structure that generated it: human conventions of grammar, stable facts about the world, common-sense regularities, and task patterns (definitions, explanations, step-by-step solutions). The network’s continual self-modification—Turing’s pupil—pushes its internal tables toward representations that make these regularities linearly separable and compositionally usable during generation.

Book Structure

This book is organized into three parts that build progressively from foundational concepts to cutting-edge applications:

- *Part I: Bayesian Foundations* — Probability as the language of uncertainty, Bayesian inference for updating beliefs, decision theory and utility for choosing actions, and reinforcement learning as sequential decision-making. This part develops the mathematical framework for reasoning under uncertainty that underlies all modern AI.
- *Part II: Statistical Learning* — Regression methods from linear to regularized, tree-based methods including random forests and gradient boosting, time series and forecasting, and model selection principles. These chapters cover the classical machine learning toolkit that remains essential for production systems.
- *Part III: Deep Learning* — Neural network fundamentals, convolutional networks for vision, sequence models and attention mechanisms, large language models, and AI agents. This part explores the architectures powering today’s most capable systems.

This unified approach brings together ideas from probability and statistics, optimization, scalable linear algebra, and high-performance computing. While deep learning initially excelled in image analysis and natural language processing, it has since expanded into diverse fields—from drug discovery to algorithmic trading—becoming a core tool for modern data scientists. Modern methodologies, software, and cloud computing make these techniques accessible to practitioners who previously relied on traditional models like generalized linear regression or tree-based methods. The ability to learn complex patterns and generate accurate predictions makes this an exciting methodology, and we hope to convey that excitement.

Course Plans

This book is targeted towards students who have completed introductory statistics and high school calculus. Basics of probability, statistics, and linear algebra are revisited as needed. We make extensive use of computational tools: R for statistical modeling and probability chapters, as well as PyTorch and JAX for deep learning. The material can be adapted for different audiences and time constraints:

Introduction to AI for MBAs (One Semester)

Chapters (theoretical sections skipped): Probability (1), Bayesian Inference (2), Decision Theory (4), A/B Testing (5), Pattern Recognition (11), Regression (12), Logistic Regression (13), Trees (14), Model Selection (16), Neural Networks up to backpropagation (18), Natural Language Processing (23).

Learning Outcomes: Evaluate AI claims critically and identify hype versus substance. Understand when machine learning adds value versus traditional analytics. Communicate effectively with data science teams about requirements and limitations. Assess and quantify uncertainty in business decisions. Make informed build-versus-buy decisions for AI capabilities.

Foundations of AI for Engineering MS (One Semester)

Chapters (theory sections optional): Probability (1), Bayesian Inference (2), Bayesian Learning (3), Decision Theory (4), A/B Testing (5), Stochastic Processes up to Markov chains (7), Reinforcement Learning (9) (briefly introduce concepts of sequential decision making), Pattern Recognition (11), Regression (12), Logistic Regression (13), Trees (14), Model Selection (16), Neural Networks (18), SGD and Optimization (20)

Learning Outcomes: Implement machine learning models from scratch using PyTorch and JAX. Derive learning algorithms mathematically and understand their convergence properties. Design AI systems for production deployment including testing and monitoring. Quantify, propagate, and communicate uncertainty throughout the modeling pipeline. Evaluate and extend state-of-the-art methods from research literature.

Part I

Bayes



1

Probability and Uncertainty

“It is remarkable that a science which began with the consideration of games of chance should have become the most important object of human knowledge...” Pierre-Simon Laplace, 1812

Probability deals with randomness and provides a language to communicate uncertainty, which is usually associated with our lack of knowledge or information. In the classical coin toss, for instance, if we knew the exact force applied, we could predict the outcome with certainty. However, practically it is never the case and we treat coin toss outcome as random.

Assigning probabilities to events is a challenging problem. Often, the probability will be applied to analyze results of experiments (observed data). Consider the coin-tossing example. Say event A represents a Head. Then, to empirically estimate probability of event A , $P(A)$, we can repeat the tosses experiment N times and count n , the number of times A occurred. The plot below shows the proportion of heads after N trials.

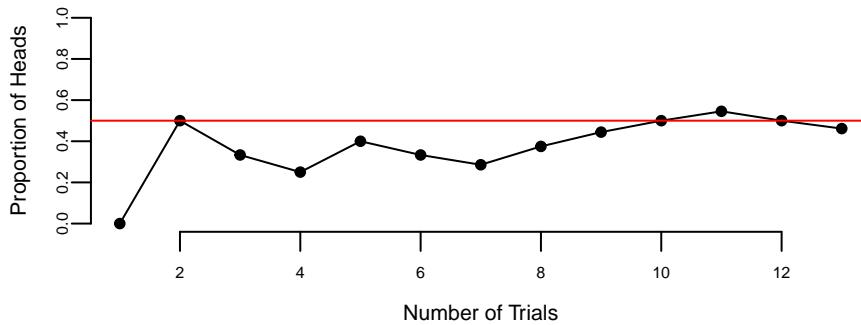


Figure 1.1: Graph of the proportion of heads after N trials

We can see that as N grows, the curve converges to 0.5. This is the law of large numbers. When N is large, n/N will be close to $P(A)$. The probability as a limit definition is natural and was proposed by von Mises.

$$P(\text{Heads}) = \lim_{N \rightarrow \infty} \frac{n}{N}.$$

However, this definition is not operational. It requires the notion of a collective, an infinite sequence of repeatable trials with random outcomes. This is an untestable assumption and was criticized by Ville (1939). On the other hand, Kolmogorov tried to operationalize probability by proposing tests for randomness in his work on algorithmic complexity theory.

We can use a relaxed definition due to Bernoulli, and define probability as simply the ratio of the number of heads to the total number of trials in a given experiment. This definition is operational and can be used to estimate the probability of an event. This definition requires the experiment to be repeated under identical conditions. If we are to repeat this experiment under different conditions, e.g. when an unbalanced coin is used, our estimate of $P(A)$ will change as well.

An alternative operational definition of probability was proposed by Frank Ramsey (1926) and later refined by Bruno de Finetti (1937). Rather than relying on long-run frequencies, this approach defines probability through the lens of rational betting behavior. The key insight is that your probability assignment for an event should correspond to the odds at which you would be willing to bet on that event.

de Finetti and Ramsey school of thought takes probability as subjective, namely personal to the observer. De Finetti famously concluded that “*Probability does not exist.*” Measuring uncertainty is personal to the observer. It’s not like mass which is a property of an object. If two different observers have differing “news” then there is an opportunity for them to bet (exchange contracts). Thus leading to an assessment of probability.

For many events most people will agree on their probabilities, for example $P(\text{Heads}) = 0.5$. In the subjective view of probability, we can measure or elicit a personal probability as a “willingness to play”. Namely, will you be willing to bet \$1 so you can get \$2 if the coin lands Tail and \$0 if Head occurs? The subjective view of probability also leads to subjective expected utility theory. You cannot separate the two. For more details, see Chapter 4.

This book primarily adopts the Bayesian (subjective) interpretation of probability, formalized through coherence (the Dutch book principle here, and de Finetti’s representation in Chapter 3). We still use frequentist language and long-run intuition as operational tools, but the organizing viewpoint is probabilistic modeling and belief updating.

Suppose you believe the probability of an event A is p . According to Ramsey’s definition, this means you should be indifferent between paying $p \cdot S$ dollars to receive S dollars if event A occurs (and nothing otherwise) and accepting $p \cdot S$ dollars to pay someone S dollars if event A occurs. However, not all probability assignments lead to rational behavior. Suppose someone assigns probabilities to events in an incoherent way. In that case, a clever adversary could construct a series of bets, called a *Dutch book*, where the individual is

guaranteed to lose money regardless of which events occur. The requirement that probabilities must be assigned in such a way that no Dutch book can be constructed against you is known as *coherence*. For a detailed derivation of probability rules using Dutch Book arguments, see the [Appendix](#).

The principle of coherence for subjective probability is the fundamental rationality constraint, in finance that would be called no-arbitrage condition Bachelier (1900).

Let us consider a simple example. Suppose you assign probability $P(A) = 0.7$ to event A occurring and probability $P(\bar{A}) = 0.2$ to event A not occurring, where \bar{A} denotes the complement of A . A Dutch book can be constructed as follows:

- Bet 1: Pay \$0.70 to receive \$1 if A occurs
- Bet 2: Pay \$0.20 to receive \$1 if \bar{A} occurs

Your total payment is \$0.90, but you will receive exactly \$1 regardless of whether A or \bar{A} occurs, for a guaranteed loss of \$0.10. The incoherence arises because $P(A) + P(\bar{A}) = 0.9 \neq 1$.

More generally, coherence implies that probabilities must satisfy the following basic properties:

1. *Non-negativity*: For any event A , we must have $P(A) \geq 0$
2. *Normalization*: For the certain event Ω (the sample space), we must have $P(\Omega) = 1$
3. *Additivity*: For mutually exclusive events A and B (i.e., $A \cap B = \emptyset$), we must have $P(A \cup B) = P(A) + P(B)$

To see why non-negativity must hold, suppose $P(A) = -0.1$ for some event A . According to the betting interpretation, you would receive \$0.10 and then pay \$1 if A occurs. If A does not occur, you keep the \$0.10 but receive nothing. However, this means you are offering to pay someone to take a bet against A —clearly irrational behavior. While standard probability theory requires non-negativity, extensions involving negative probabilities have been explored in fields like physics and quantum computing, as well as in Bayesian modeling as mixing distributions for unobserved latent variables (Nick Polson and Sokolov 2025).

The normalization requirement $P(\Omega) = 1$ ensures that you assign probability one to something that is certain to happen. If $P(\Omega) < 1$, you would be willing to pay less than \$1 to receive \$1 with certainty, allowing an arbitrageur to make a riskless profit. Conversely, if $P(\Omega) > 1$, you would pay more than \$1 for a certain payoff of \$1, guaranteeing a loss.

The additivity axiom ensures consistency across mutually exclusive events. If A and B cannot both occur, then betting on “ A or B ” should cost the same

as placing separate bets on A and on B . Violating this principle again opens the door to Dutch books.

Another axiom $P(A) + P(\bar{A}) = 1$ just follows from the normalization requirement and the additivity axiom.

These are precisely the axioms proposed by Andrey Kolmogorov (1933) in his foundational work on probability theory. They provide mathematical structure to probability. Although, Kolmogorov's axioms are agnostic to any definition of probability and are purely mathematical in nature, the fact that they can be derived from the Dutch book argument shows their applicability to rational decision-making under uncertainty. Any violation of these axioms opens you to guaranteed losses through carefully constructed bets. However, Kolmogorov's axiomatic approach helps us to derive results in more complex settings. For examples, Kolmogorov's framework is applicable in infinite sample spaces and continuous random variables.

The axioms provide a number of rules that probabilities must follow. There are several important corollaries that can help us assign probabilities to events. Here are some important corollaries that follow from the Kolmogorov axioms:

1. *Complement rule:* Let “not A ” denote the complement of event A .

$$P(\text{not } A) = 1 - P(A).$$

2. *Monotonicity:* If $A \subset B$, then $P(A) \leq P(B)$. In other words, the probability of a larger set is greater than or equal to the probability of a subset.
3. *Subadditivity:* This is a generalization of the addition rule, where the equality holds when events A and B are mutually exclusive.

$$P(A \text{ or } B) \leq P(A) + P(B).$$

4. *Inclusion–exclusion principle:* This principle extends subadditivity to the case where A and B are not necessarily mutually exclusive.

$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B).$$

5. *Conditional probability:* The conditional probability of A given B is

$$P(A | B) = \frac{P(A \text{ and } B)}{P(B)}.$$

6. *Bayes rule* simply reverses the conditioning to compute $P(A | B)$ from $P(B | A)$ —a disciplined probability accounting.

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}.$$

7. *Law of total probability* is a direct consequence of the definition of conditional probability and the normalization axiom. It states that if B_1, B_2, \dots, B_n are mutually exclusive and exhaustive events, then

$$P(A) = \sum_{i=1}^n P(A \text{ and } B_i) = \sum_{i=1}^n P(A | B_i)P(B_i).$$

All of these axioms follow simply from the principle of coherence and the avoidance of Dutch book arbitrage. This includes the Bayes rule itself (de Finetti 1937; Shimony 1955). If there is arbitrage present in the market, it should be “traded”. It often happens when subjective probabilities do not match.

Bayes rule is a fundamental rule of probability that allows us to calculate conditional probabilities. It is a direct consequence of the definition of conditional probability and the normalization axiom. This rule will become central to learning and inference in artificial intelligence.

Bayes rule simply provides a disciplined probability accounting of how probabilities get updated in light of evidence. A rational agent requires that their subjective probabilities must obey the principle of coherence. Namely in announcing the set of probabilities they cannot undergo a sure loss. Interestingly enough, this is enough to provide a similar framework to the axiomatic approach of Kolmogorov.

These corollaries and principles help in deriving further results and provide additional tools for analyzing and understanding probability and random processes based on the fundamental principles laid out by Kolmogorov. Arguably the most important rule is Bayes rule for conditional probability.

The rise of artificial intelligence has definitively established Bayesian inference as a cornerstone of modern learning algorithms. One of the key properties of probabilities is that they are updated as you learn new information. Conditional means given its personal characteristics or the personal situation. Personalization algorithms used by many online services rely on this concept. One can argue that all probabilities are conditional in some way. The process of Bayesian updating is central to how machines learn from observed data. Rational human behavior ought to adhere to Bayes rule, although there is much literature documenting the contrary.

1.1 Odds as Probabilities

Another way, sometimes more convenient, to talk about uncertainty and to express probabilities via odds, such as 9 to 2 or 3 to 1. Odds express the ratio

of favorable to unfavorable outcomes (Success:Failure), while probability is the chance of an event happening out of all possibilities (Success / Total).

We assign odds “on A ” (or “in favor of A ”) versus odds “against A ”. For example, if the probability of a Chicago Bears’ Super Bowl win is $P(A) = 2/11$, the odds against them are $(1 - 2/11)/(2/11) = 9/2$, or “9 to 2”. This means for every 2 times they win, they lose 9 times. Conventionally, “odds on” often refers to the reverse, but to avoid ambiguity, we will speak of probability or odds against.

$$O(A) = \frac{P(\bar{A})}{P(A)} = \frac{1 - P(A)}{P(A)}$$

Equivalently, probabilities can be determined from odds

$$P(A) = \frac{1}{1 + O(A)}$$

For example, if the odds are one $O(A) = 1$, then for every \$1 bet you will pay out \$1. This event has probability 0.5.

If $O(A) = 2$, then you are willing to offer 2 : 1. For a \$1 bet you’ll payback \$3. In terms of probability $P(A) = 1/3$.

Odds are primarily used in betting markets. For example, let’s re-analyze the 2016 election in the US.

Example 1.1 (Odds). One of the main sources of prediction markets is bookmakers who take bets on outcomes of events (mostly sporting) at agreed upon odds. Figure 1.2 shows the odds used by several bookmakers to take bets on the winner of the US presidential election in 2016. At that time the market was predicting that Hillary Clinton would win over Donald Trump, the second favorite, with odds 7/3. The table is generated by the Oddschecker website.

	Sign Up	FREE BETS	£200	£20	£30	£50	£30	£50	£50	£60	£30	£20	£30	£20	£20	£200	£30	£30	£20	£30	£30	£100	£200	£20	£25	£20	£25	£30
Sort By:	Favourite																											
Hillary Clinton	+		2/7	1/4	3/10	2/7	3/10	1/4	1/4	2/7	1/4	3/10	2/7	2/7	2/7	1/4	2/9	2/9	1/4	3/10	1/4	3/10	3/11	1/4	3/10	1/3	3/10	1/3
Donald Trump	+		11/4	11/4	5/2	11/4	13/5	3	11/4	5/2	11/4	13/5	11/4	11/4	11/4	11/4	3	3	11/4	13/5	3	13/5	56/19	13/5	3	3	3	3
Bernie Sanders	+		66	66	66	66	66	66	66	66	66	66	50	66	66	66	66	66	66	66	66	66	104	86	59			

Figure 1.2: Presidential Odds 2016

Ahead of time we can assign probabilities of winning to each candidate. According to the bookmakers’ odds the candidate with highest chance to win is Hillary Clinton. The best odds on Clinton are 1/3; this means that you have to risk \$3 to win \$1 offered by Matchbook. Odds dynamically change as new information arrives. There is also competition between the Bookmakers and the Market is adapting to provide the best possible odds. Ladbrokes is the largest UK bookie and Betfair is an online exchange. A bookmaker sets

their odds trying to get equal public action on both sides, otherwise they are risking to stay out of business.

Example 1.2 (Kentucky Derby). The Kentucky Derby happens once a year – first Saturday in May. In horse racing the odds are set by the betting public. The racetrack collects all the bets, takes a fee (18%), and then redistributes the pool to the winning tickets. The race is $1\frac{1}{4}$ miles (2 kilometers) and is the first time the three-year old horses have raced the distance.

There was a long period where favorites rarely won. Only six favorites have won in the 36 year period from 1979 to 2013. Recently favorites have won many times in a row. The market is getting better at predicting who's going to win. Here's the data

Horse Name	Year	Odds
Spectacular Bid	1979	3/5
Fusaichi Pegasus	2000	2.3/1
Street Sense	2007	9/2
Big Brown	2008	5/2

Recently, favorites have had a lot more success

Horse Name	Year	Odds
California Chrome	2014	5/2
American Pharoah	2015	2/1
Nyquist	2016	3.3/1
Always Dreaming	2017	5.2/1

The most famous favorite to win is Secretariat (1973) who won with odds 3/2 in a record time of 1 minute 59 and 2/5 seconds. Monarchos was the only other horse that in 2001 has broken two minutes at odds 11.5/1.

Example 1.3 (Exacta Betting and the Harville Formula). How can probability help you with betting on the race? There are many different types of bets, and probability can help you find *fair* odds. The Derby is a Grade 1 stakes race for three-year-old thoroughbred horses. Colts and geldings carry 126 pounds and fillies 121. The odds are set by pari-mutuel betting by the public. After all the wagers have been placed, the racetrack takes a fee (18%). After the winning horse passes the finishing line, the pool of money is redistributed to the winning tickets. Random betting therefore loses you 18%, so it's important to learn some empirical facts to try and tilt the odds in your favor.

For example, you can place bets as follows:

- *Win*: “\$2 win horse 1”
- *Straight Exacta*: “\$2 exacta 1 with 2”
- *Exacta Box*: “\$2 exacta box 1 and 2” You win with *either* order: 2 bets = \$4.

Consider a hypothetical race where *Sovereignty* wins at 9/1 odds and *Journalism* comes second at 7/2 odds. For a \$2 bet on *Sovereignty* to Win at 9/1, the payout would be $2 \cdot 9 + 2 = \$20$ (the 9/1 win plus your initial \$2 bet returned).

Let's figure out the *fair* value for an exacta bet given that you know the win odds. This is known as the *Harville formula*. The exacta is probably one of the most popular bets for many horseplayers, corresponding to predicting the first two horses in the correct order.

The Harville formula provides an answer. We use the rule of conditional probability. The probability for the straight exacta of horses *A* beating horse *B* is:

$$P(A \text{ beats } B) = P(A \text{ Wins}) \cdot P(B \text{ Second} \mid A \text{ Wins})$$

A reasonable assessment of $P(B \text{ Second} \mid A \text{ Wins})$ can be derived as follows. Renormalizing the probabilities by removing the winner *A* and distributing the probability mass to the remaining horses gives:

$$P(B \text{ Second} \mid A \text{ Wins}) = \frac{P(B \text{ Wins})}{1 - P(A \text{ Wins})}$$

In total, the fair price for the exacta is:

$$P(A \text{ beats } B) = P(A \text{ Wins}) \cdot \frac{P(B \text{ Wins})}{1 - P(A \text{ Wins})}$$

Therefore, we have:

$$p_{12} = p_1 \cdot \frac{p_2}{1 - p_1} \text{ where } p_1 = \frac{1}{1 + O_1}, p_2 = \frac{1}{1 + O_2}$$

Solving for odds, we get the *Harville formula*:

$$O_{12} = O_1(1 + O_2) - 1$$

Using our example with 9/1 and 7/2 odds: $O_{12} = 9 \cdot (1 + 3.5) - 1 = 39.5/1$.

Notice that the actual payout is determined solely by the volume of money wagered on that combination. There's no requirement it matches our probabilistic analysis. However, the Harville formula gives us an idea of fair value.

Some bettors searching for value try to find significantly undervalued exacta bets relative to the Harville formula.

There are many other factors to consider: jockey performance, bloodlines, and post positions can all matter significantly in determining the actual race outcome.

Example 1.4 (Boy-Girl Paradox). If a woman has two children and one is a girl, the chance that the other child is also female has to be 50 – 50, right? But it's not. Let's list the possibilities of girl-girl, girl-boy and boy-girl. So the chance that both children are girls is 33 percent. Once we are told that one child is female, this extra information constrains the odds. (Even weirder, the author demonstrates that the odds change again if we're told that one of the girls is named Florida.) In terms of conditional probability, the four possible combinations are

$$BB \ BG \ GB \ GG$$

Conditional on the information that one is a girl means that you know we can't have the *BB* scenario. Hence we are left with three possibilities

$$BG \ GB \ GG$$

In one of these is the other a girl. Hence 1/3.

It's a different question if we say that the first child is a girl. Then the probability that the other is a girl is 1/2 as there are two possibilities

$$GB \ GG$$

This leads to the probability of 1/2.

Example 1.5 (Galton Paradox). You flip three fair coins. What is the $P(\text{all alike})$?

Assuming a fair coin (i.e. $P(H) = P(T) = 1/2$), a formal approach might consist of computing the probability for all heads or all tails, which is

$$\begin{aligned} P(HHH) &\equiv P(H \text{ and } H \text{ and } H) \\ &= P(H) \times P(H) \times P(H) \\ &= \left(\frac{1}{2}\right)^3 \end{aligned}$$

and, since we're ultimately interested in the probability of either (mutually exclusive) case,

$$\begin{aligned} P(\text{all alike}) &= P(HHH \text{ or } TTT) \\ &= P(HHH) + P(TTT) \\ &= 2 \times \frac{1}{8} \end{aligned}$$

One could arrive at the same conclusion by enumerating the entire sample space and counting the events. Now, what about a simpler argument like the following. In a run of three coin flips, two coins will always share the same result, so the probability that the “remaining/last” coin matches the other two is $1/2$; thus,

$$P(\text{all alike}) = 1/2$$

There are 8 equally likely outcomes. Two are ‘all alike’ (HHH, TTT). So $2/8 = 1/4$. The error in reasoning is assuming that ‘two must correspond’ fixes the first two coins, but ‘two alike’ could be coins 1&2, 2&3, or 1&3.

For a real treatment of the subject, we highly recommend reading Galton’s essay at galton.org.

Example 1.6 (Three Cards). Suppose that you have three cards: one red/red, one red/blue and one blue/blue. You randomly draw a card and place it face down on a table and then you reveal the top side. You see that it’s red. What’s the probability the other side is red? $1/2$? No, it’s $2/3$! By a similar logic there are six initial possibilities

$$B_1B_2 \ B_2B_1 \ BR \ RB \ R_1R_2 \ R_2R_1$$

where 1 and 2 index the sides of the same colored cards.

If we now condition on the top side being red we see that there are still three possibilities left

$$RB \ R_1R_2 \ R_2R_1$$

Hence the probability is $2/3$ and not the intuitive $1/2$.

Example 1.7 (NFL: New England Patriots Coin Toss). Let’s consider another example and calculate the probability of winning 19 coin tosses out of 25. The NFL team New England Patriots won 19 out of 25 coin tosses in the 2014-15 season. What is the probability of this happening?

Let X be a random variable equal to 1 if the Patriots win and 0 otherwise. It’s reasonable to assume $P(X = 1) = \frac{1}{2}$. The probability of observing the sequence in which there is 1 on the first 19 positions and 0 afterwards is $(1/2)^{25}$. We can code a typical sequence as,

$$1, 1, 1, \dots, 1, 0, 0, \dots, 0.$$

There are 177,100 different sequences of 25 games where the Patriots win 19. There are $25! = 1 \cdot 2 \cdot \dots \cdot 25$ ways to re-arrange this sequence of zeroes and ones. Further, all zeroes and ones are interchangeable and there are $19!$ ways to re-arrange the ones and $6!$ ways to rearrange the sequence of zeroes. Thus, the total number of different winning sequences is

```
factorial(25) / (factorial(19) * factorial(25 - 19))
## 177100
```

Each potential sequence has probability 0.5^{25} , thus

$$P(\text{Patriots win 19 out of 25 tosses}) = 177,100 \times 0.5^{25} = 0.005$$

Often, it is easier to communicate uncertainties in the form of odds. In terms of betting odds of $1 : 1$ gives $P = \frac{1}{2}$, odds of $2 : 1$ (I give 2 for each 1 you bet) is $P = \frac{1}{3}$.

Remember, odds, $O(A)$, is the ratio of the probability of happening over not happening,

$$O(A) = (1 - P(A))/P(A),$$

equivalently,

$$P(A) = \frac{1}{1 + O(A)}.$$

The odds of the Patriots winning sequence are then 1 to 199

```
0.005 / (1 - 0.005)
## 0.005
```

Example 1.8 (Hitting Streak). Pete Rose of the Cincinnati Reds set a National League record of hitting safely in 44 consecutive games. How likely is such a long sequence of safe hits to be observed? If you were a bookmaker, what odds would you offer on such an event? This means that he safely reached first base after hitting the ball into fair territory, without the benefit of an error or a fielder's choice at least once in every one of those 44 games. Here are a couple of facts we know about him:

1. Rose was a 300 hitter, he hits safely 3 times out of 10 attempts
2. Each at bat is assumed to be independent, i.e., the current at bat doesn't affect the outcome of the next.

Assuming he comes to bat 4 times each game, *what probability might reasonably be associated with that hitting streak?* First we define notation. We use A_i to denote an event of hitting safely at game i , then

$$\begin{aligned} P(\text{Rose Hits Safely in 44 consecutive games}) &= \\ P(A_1 \text{ and } A_2 \dots \text{ and } A_{44}) &= P(A_1)P(A_2) \dots P(A_{44}) \end{aligned}$$

We now need to find $P(A_i)$ s where $P(A_i) = 1 - P(\text{not } A_i)$

$$\begin{aligned} P(A_1) &= 1 - P(\text{not } A_1) \\ &= 1 - P(\text{Rose makes 4 outs}) \\ &= 1 - (0.7)^4 = 0.76 \end{aligned}$$

For the winning streak, then we have $(0.76)^{44} = 0.0000057$, a very low probability. In terms of odds, there are three basic inferences

1. This means that the odds for a particular player as good as Pete Rose starting a hitting streak today are 175,470 to 1.
2. This doesn't mean that the run of 44 won't be beaten by some player at some time: the Law of Very Large Numbers
3. Joe DiMaggio's record is 56. He is a 325 hitter, thus we have $(0.792)^{56} = 2.13 \times 10^{-6}$ or 455,962 to 1. It's going to be hard to beat.

The independence assumption underlying this calculation does not account for the popular belief in the "hot hand"—the idea that a player who has been successful recently is more likely to succeed again.

Example 1.9 (Derek Jeter). Sample averages can have paradoxical behavior. This is related to the field of causation and the property of confounding. Let's compare Derek Jeter and David Justice batting averages. In both 1995 and 1996, Justice had a higher batting average than Jeter did. However, when you combine the two seasons, Jeter shows a higher batting average than Justice! This is just a property of averages and finer subset selection can change your average effects.

	1995	1996		Combined	
Derek Jeter	12/48	0.250	183/582	0.314	195/630
David Justice	104/411	0.253	45/140	0.321	149/551

This situation is known as *confounding*. It occurs when two separate and different populations are aggregated to give misleading conclusions. The example shows that if A, B, C are events it is possible to have the three inequalities

$$\begin{aligned} P(A | B \text{ and } C) &> P(A | B \text{ and } \bar{C}) \\ P(A | \bar{B} \text{ and } C) &> P(A | \bar{B} \text{ and } \bar{C}) \\ P(A | C) &< P(A | \bar{C}) \end{aligned}$$

The three inequalities can't hold simultaneously when $P(B | C) = P(B | \bar{C})$.

Example 1.10 (Birthday Problem). The birthday problem (Persi Diaconis and and Mosteller 1989) is a classic problem in probability theory that explores the counterintuitive likelihood of shared birthdays within a group. Surprisingly, in a room of 23 people, the probability of shared birthdays is 50%. With 70 people, the probability is 99.9%.

In general, given N items (people) randomly distributed into c categories (birthdays), where the number of items is small compared to the number of categories $N \ll c$, the probability of no match is given by

$$P(\text{no match}) \approx \exp(-N^2/2c).$$

Given A_i is the event that person i has a matching birthday with someone, we have

$$P(\text{no match}) = \prod_{i=1}^{N-1} (1 - P(A_i)) = \exp\left(\sum_{i=1}^{N-1} \log(1 - P(A_i))\right).$$

Here $P(A_i) = \frac{i}{c}$. Then use the approximation $\log(1 - x) \approx -x$ for small x to get $P(\text{no match})$.

$$\sum_{i=1}^{N-1} \log(1 - P(A_i)) \approx -\sum_{i=1}^{N-1} \frac{i}{c} = -\frac{N(N-1)}{2c}.$$

The probability of at least two people sharing a birthday is then the complement of the probability above:

$$P(\text{At least one shared birthday}) = 1 - P(\text{no match}).$$

Solving for $P(\text{match}) = 1/2$, leads to a square root law $N = 1.2\sqrt{c}$, if $c = 365$ then $N = 23$, and if $c = 121$ (near birthday match), then $N = 13$.

The unintuitive nature of this result is a consequence of the fact that there are many potential pairs of people in the group, and the probability of at least one pair sharing a birthday increases quickly as more people are added. The birthday problem is often used to illustrate concepts in probability, combinatorics, and statistical reasoning. It's a great example of how our intuitions about probabilities can be quite different from the actual mathematical probabilities.

1.2 Random Variables: Quantifying Uncertainty

A random variable is a function that maps the outcomes of a random experiment (events) to real numbers. It essentially assigns a numerical value to

each outcome in the sample space of a random experiment. In other words, a random variable provides a bridge between the abstract concept of events in a sample space and the concrete calculations involving numerical values and probabilities. Similar to assigning probabilities to events, we can assign respective probabilities to random variables.

For example, consider a random experiment of rolling a die. Here, an event could be “the outcome is an even number”, and the random variable could be the actual number that shows up on the die. The probability of the event “the outcome is an even number” is 0.5, and the probability distribution of the random variable is a list of all numbers from 1 to 6 each with a probability of 1/6.

While events and random variables are distinct concepts, they are closely related through the framework of probability theory, with random variables serving as a key tool for calculating and working with probabilities of events.

Random variables are quantities that we are not certain about. A random variable that can take a finite or a countable number of values is called a *discrete random variable* (number of rainy days next week). Otherwise, it will be a *continuous random variable* (amount of rain tomorrow).

Discrete random variables are often constructed by assigning specific values to events such as $\{X = x\}$ which corresponds to the outcomes where X equals a specific number x . For example

1. Will a user click-through on a Google ad? (0 or 1)
2. Who will win the next presidential election? (Republican=1, Democrat=2, Independent=3)

To fix notation, we will use $P(X = x)$ to denote the probability that random variable X is equal to x . A map from all possible values x of a discrete random variable X to probabilities is called a *probability mass function* $p(x)$. We will interchangeably use $P(X = x)$ and $p(x)$. An important property of the probability mass function is that (normalization Kolmogorov axiom)

$$\sum_{x \in S} p(x) = 1.$$

Here S denotes the set of all possible values of random variable X .

Clearly, all probabilities have to be greater than or equal to zero, so that $p(x) \geq 0$.

Often, we are interested in

$$F(x) = P(X \leq x) = \sum_{y \leq x} p(y),$$

this is the cumulative distribution function (CDF).

The CDF is a monotonically increasing function (never decreases as x increases). In other words, if $a \leq b$, then $F_X(a) \leq F_X(b)$. The value of the CDF always lies between 0 and 1, inclusive.

Example 1.11 (Discrete CDF). Suppose X is a discrete random variable that represents the outcome of rolling a six-sided die. The probability mass function (PMF) of X is:

$$P(X = x) = \frac{1}{6}$$

for $x = 1, 2, 3, 4, 5, 6$

The CDF of X , $F(x)$, is calculated as follows:

- For $x < 1$, $F(x) = 0$ (since it's impossible to roll less than 1).
- For $1 \leq x < 2$, $F(x) = \frac{1}{6}$ (the probability of rolling a 1).
- For $2 \leq x < 3$, $F(x) = \frac{1}{6} + \frac{1}{6} = \frac{2}{6}$ (the probability of rolling a 1 or 2).
- This pattern continues, adding $\frac{1}{6}$ for each integer interval up to 6.
- For $x \geq 6$, $F(x) = 1$ (since it's certain to roll a number 6 or less).

Graphically, the CDF of a discrete random variable is a step function that increases at the value of each possible outcome. It's flat between these outcomes because a discrete random variable can only take specific, distinct values.

```
plot(ecdf(1:6), main = "")
```

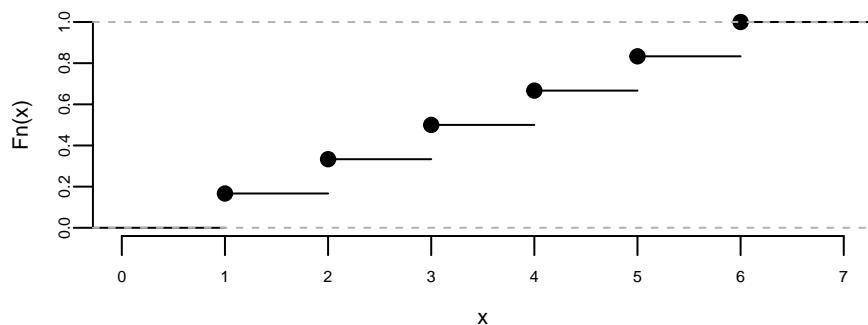


Figure 1.3: CDF of a discrete random variable

1.2.1 Continuous Random Variables

If we want to build a probabilistic model of a stock price or return, we need to use a continuous random variable that can take an interval of values. Instead

of a frequency function we will use a *density function*, $p(x)$ to describe a continuous variable. Unlike the discrete case, $p(x)$ is not the probability that the random variable takes value x . Rather, we need to talk about the value being inside an interval. For example, the probability of X with density $p(x)$ being inside any interval $[a, b]$, with $a < b$ is given by

$$P(a < X < b) = \int_a^b p(x)dx.$$

The total probability is one as $\int_{-\infty}^{\infty} p(x)dx = 1$. The simplest continuous random variable is the uniform. A uniform distribution describes a variable which takes on any value as likely as any other. For example, if you are asked about what would be the temperature in Chicago on July 4 of next year, you might say anywhere between 20 and 30 C. The density function of the corresponding uniform distribution is then

$$p(x) = \begin{cases} 1/10, & 20 \leq x \leq 30 \\ 0, & \text{otherwise} \end{cases}$$

Under this model, the probability of temperature being between 25 and 27 degrees is

$$P(25 \leq x \leq 27) = \int_{25}^{27} p(x)dx = (27 - 25)/10 = 0.2$$

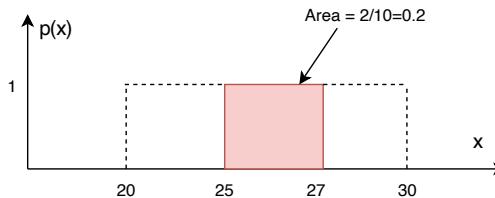


Figure 1.4: Uniform Distribution: Probability of temperature being between 25 and 27

The Cumulative Distribution Function for a continuous random variable, it is defined similarly to discrete RV CDF as

$$F(x) = P(X \leq x) = \int_{-\infty}^x p(t)dt$$

It is a non-decreasing function and takes values in [0,1].

Example 1.12 (Continuous CDF for Uniform Distribution).

$$p(x) = \begin{cases} 1 & \text{if } 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

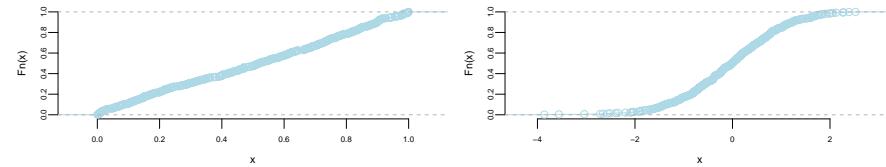
The CDF, $F(x)$, is obtained by integrating the PDF:

- For $x < 0$, $F(x) = 0$.
- For $0 \leq x \leq 1$, $F(x) = \int_0^x 1 dt = x$.
- For $x > 1$, $F(x) = 1$.

So, the CDF of this uniform distribution is a linear function that increases from 0 to 1 as x goes from 0 to 1.

Graphically, the CDF of a continuous random variable is a smooth curve. It starts at 0, increases as x increases, and eventually reaches 1. The exact shape of the curve depends on the distribution of the variable, but the smooth, non-decreasing nature is a common feature. Figure below shows the CDF of a uniform and normal random variable, respectively.

```
plot(ecdf(runif(500)), main = "", col = "lightblue", pch = 21,
      bg = "grey")
plot(ecdf(rnorm(500)), main = "", col = "lightblue", pch = 21,
      bg = "grey")
```



(a) CDF of a uniform random variable (a) CDF of a normal random variable

1.2.2 The Inverse CDF Method

The inverse distribution method uses samples of uniform random variables to generate draws from random variables with a continuous distribution function, F . Since $F(x)$ is uniformly distributed on $[0, 1]$, draw a uniform random variable and invert the CDF to get a draw from F . Thus, to sample from F ,

Step 1: Draw $U \sim U[0, 1]$
 Step 2: Set $X = F^{-1}(U)$,

where $F^{-1}(U) = \inf\{x : F(x) = U\}$.

This inversion method provides i.i.d. draws from F provided that $F^{-1}(U)$ can be exactly calculated. For example, the CDF of an exponential random variable with parameter μ is $F(x) = 1 - \exp(-\mu x)$, which can easily be inverted. When F^{-1} cannot be analytically calculated, approximate inversions can be used. For example, suppose that the density is a known analytical function. Then, $F(x)$ can be computed to an arbitrary degree of accuracy on a grid and inversions can be approximately calculated, generating an approximate draw from F . With all approximations, there is a natural trade-off between computational speed and accuracy. One example where efficient approximations are possible are inversions involving normal distributions, which is useful for generating truncated normal random variables. Outside of these limited cases, the inverse transform method does not provide a computationally attractive approach for drawing random variables from a given distribution function. In particular, it does not work well in multiple dimensions.

1.2.3 Functional Transformations

The second main method uses functional transformations to express the distribution of a random variable that is a known function of another random variable. Suppose that $X \sim F$, admitting a density f , and that $y = h(x)$ is an increasing continuous function. Thus, we can define $x = h^{-1}(y)$ as the inverse of the function h . The distribution of y is given by

$$F_Y(y) = P(Y \leq y) = \int_{-\infty}^{h^{-1}(y)} f(x) dx = F_X(X \leq h^{-1}(y)).$$

Differentiating with respect to y gives the density via Leibnitz's rule:

$$f_Y(y) = f(h^{-1}(y)) \left| \frac{d}{dy}(h^{-1}(y)) \right|,$$

where we make explicit that the density is over the random variable Y . This result is used widely. For example, if $X \sim \mathcal{N}(0, 1)$, then $Y = \mu + \sigma X$. Since $x = h^{-1}(y) = \frac{y-\mu}{\sigma}$, the distribution function is $F(\frac{x-\mu}{\sigma})$ and density

$$f_Y(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2\right).$$

Transformations are widely used to simulate both univariate and multivariate random variables. As examples, if $Y \sim \chi^2(\nu)$ and ν is an integer, then $Y = \sum_{i=1}^{\nu} X_i^2$ where each X_i is independent standard normal. Exponential random variables can be used to simulate χ^2 , Gamma, Beta, and Poisson random variables. The famous Box-Muller algorithm simulates normals from uniform and exponential random variables. In the multivariate setting, Wishart (and inverse Wishart) random variables can be simulated via sums of squared vectors of standard normal random variables.

1.3 Expectation and Variance (Reward and Risk)

An expected value of a random variable, denoted by $E(X)$ is a weighted average. Each possible value of a random variable is weighted by its probability. For example, Google Maps uses expected value when calculating travel times. We might compute two different routes by their expected travel time. Typically, a forecast or expected value is all that is required — these expected values can be updated in real time as we travel. Say I am interested in travel time from Washington National airport to Fairfax in Virginia. The histogram below shows the travel times observed for a work day evening and were obtained from [Uber](#).

Example 1.13 (Uber). Let's look at the histogram of travel times from Fairfax, VA to Washington, DC

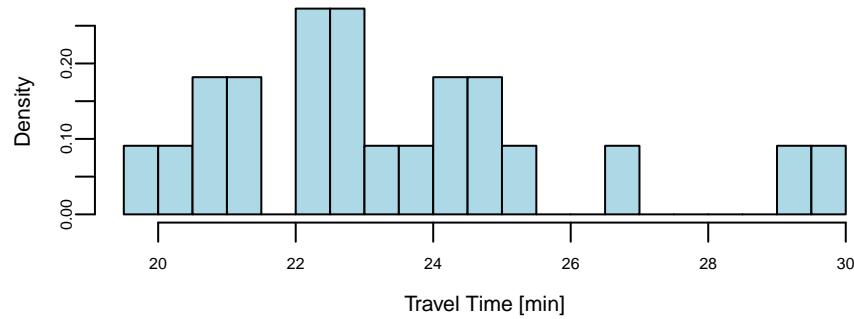


Figure 1.7: Travel times in the evening

From this dataset, we can empirically estimate the probabilities of observing different values of travel times

Travel Time	Probability
18	0.05
22	0.77
28	0.18

There is a small chance (5%) I can get to Washington, DC in 18 minutes, which probably happens on a holiday and a non-trivial chance (18%) to travel for 28 minutes, possibly due to a sports game or bad weather. Most of the time (77%) our travel time is 22 minutes. However, when Uber shows you the travel

time, it uses the expected value as a forecast rather than the full distribution. Specifically, you will be given an expected travel time of 23 minutes.

```
0.05 * 18 + 0.77 * 22 + 0.18 * 28
## 23
```

It is a simple summary that takes into account travel accidents and other events that can affect travel time as best as it can.

The expected value $E(X)$ of discrete random variable X which takes possible values $\{x_1, \dots, x_n\}$ is calculated using

$$E(X) = \sum_{i=1}^n x_i P(X = x_i)$$

For example, in a binary scenario, if $X \in \{0, 1\}$ and $P(X = 1) = p$, then $E(X) = 0 \times (1 - p) + 1 \times p = p$. The expected value of a Bernoulli random variable is simply the probability of success. In many binary scenarios, a probabilistic forecast is sufficient.

If X is continuous with probability distribution $p(x)$, then we have to calculate the expectation as an integral

$$E(X) = \int xp(x)dx \text{ and } \int p(x)dx = 1.$$

1.3.1 Standard Deviation and Covariance

Variance measures the spread of a random variable around its expected value $\mu = E(X)$. For a discrete random variable X with possible values $\{x_1, \dots, x_N\}$, we have

$$\text{Var}(X) = E((X - \mu)^2) = \sum_{i=1}^N (x_i - \mu)^2 P(X = x_i).$$

In the continuous case, we have

$$\text{Var}(X) = \int_{-\infty}^{\infty} (x - \mu)^2 p(x)dx, \text{ where } \mu = E(X) = \int_{-\infty}^{\infty} p_X(x)dx.$$

The standard deviation is more convenient and is the square root of variance $\text{sd}(X) = \sqrt{\text{Var}(X)}$. Standard deviation has the desirable property that it is measured in the same units as the random variable X itself and is a more useful measure.

Suppose that we have two random variables X and Y . We need to measure whether they move together or in opposite directions. The *covariance* is defined by

$$\text{Cov}(X, Y) = E([(X - E(X))(Y - E(Y))]).$$

When X and Y are discrete and we are given the joint probability distribution, we need to calculate

$$\text{Cov}(X, Y) = \sum_{x,y} (x - E(X))(y - E(Y))p(x, y).$$

Covariance is measured in units of $X \times$ units of Y . This can be inconvenient and makes it hard to compare covariances of different pairs of variables. A more convenient metric is the *correlation*, which is defined by

$$\text{Cor}(X, Y) = \frac{\text{Cov}(X, Y)}{\text{sd}(X)\text{sd}(Y)}.$$

Correlation, $\text{Cor}(X, Y)$, is unitless and takes values between -1 and 1.

In the case of joint continuous distribution it is convenient to use the covariance matrix Σ which is defined as

$$\Sigma = \begin{bmatrix} \text{Var}(X) & \text{Cov}(X, Y) \\ \text{Cov}(X, Y) & \text{Var}(Y) \end{bmatrix}.$$

If X and Y are independent, then $\text{Cov}(X, Y) = 0$ and Σ is diagonal. The correlation matrix is defined as

$$\rho = \begin{bmatrix} 1 & \text{Cor}(X, Y) \\ \text{Cor}(X, Y) & 1 \end{bmatrix}.$$

If X and Y have an exact linear relationship, then $\text{Cor}(X, Y) = 1$ and $\text{Cov}(X, Y)$ is the product of standard deviations. In matrix notation, the relation between the covariance matrix and correlation matrix is given by

$$\rho = \text{diag}(\Sigma)^{-1/2} \Sigma \text{diag}(\Sigma)^{-1/2},$$

where Σ is a diagonal matrix with standard deviations on the diagonal.

1.3.2 Portfolios: linear combinations

Calculating means and standard deviations of combinations of random variables is a central tool in probability. It is known as the portfolio problem. Let P be your portfolio, which comprises a mix of two assets X and Y , typically stocks and bonds,

$$P = aX + bY,$$

where a and b are the portfolio weights, typically $a + b = 1$, as we are allocating our total capital. Imagine that you have placed a dollars on the random outcome X , and b dollars on Y . The portfolio P measures your total weighted outcome.

Key portfolio rules: The expected value and variance follow the relations

$$\begin{aligned} E(aX + bY) &= aE(X) + bE(Y) \\ \text{Var}(aX + bY) &= a^2\text{Var}(X) + b^2\text{Var}(Y) + 2ab\text{Cov}(X, Y), \end{aligned}$$

with *covariance* defined by

$$\text{Cov}(X, Y) = E((X - E(X))(Y - E(Y))).$$

Expectation and variance help us to understand the long-run behavior. When we make long-term decisions, we need to use the expectations to avoid biases.

The covariance is related to the correlation by $\text{Cov}(X, Y) = \text{Corr}(X, Y) \cdot \sqrt{\text{Var}(X) \cdot \text{Var}(Y)}$.

Example 1.14 (Tortoise and Hare). Tortoise and Hare are selling cars. Say X is the number of cars sold and probability distributions, means and variances are given by the following table

	X				Mean	Variance	sd
	0	1	2	3	$E(X)$	$\text{Var}(X)$	$\sqrt{\text{Var}(X)}$
Tortoise	0	0.5	0.5	0	1.5	0.25	0.5
Hare	0.5	0	0	0.5	1.5	2.25	1.5

Let's calculate Tortoise's expectations and variances

$$\begin{aligned} E(T) &= (1/2)(1) + (1/2)(2) = 1.5 \\ \text{Var}(T) &= E(T^2) - E(T)^2 \\ &= (1/2)(1)^2 + (1/2)(2)^2 - (1.5)^2 = 0.25 \end{aligned}$$

Now the Hare's

$$\begin{aligned} E(H) &= (1/2)(0) + (1/2)(3) = 1.5 \\ \text{Var}(H) &= (1/2)(0)^2 + (1/2)(3)^2 - (1.5)^2 = 2.25 \end{aligned}$$

What do these tell us about the long run behavior?

1. Tortoise and Hare have the same expected number of cars sold.
2. Tortoise is more predictable than Hare. He has a smaller variance.

The standard deviations $\sqrt{\text{Var}(X)}$ are 0.5 and 1.5, respectively. Given two equal means, you always want to pick the lower variance. If we are to invest in one of those, we prefer Tortoise.

What about a portfolio of Tortoise and Hare? Suppose I want to evenly split my investment between Tortoise and Hare. What is the expected number of cars sold and the variance of the number of cars sold?

$$\mathbb{E}\left(\frac{1}{2}T + \frac{1}{2}H\right) = \frac{1}{2}\mathbb{E}(T) + \frac{1}{2}\mathbb{E}(H) = 1.5$$

For variance, we need to know $\text{Cov}(T, H)$. Let's take $\text{Cov}(T, H) = -1$ and see what happens.

$$\begin{aligned}\text{Var}\left(\frac{1}{2}T + \frac{1}{2}H\right) &= \frac{1}{4}\text{Var}(T) + \frac{1}{4}\text{Var}(H) + \frac{1}{2}\text{Cov}(T, H) \\ &= 0.0625 + 0.5625 - 0.5 = 0.125\end{aligned}$$

Notice that the portfolio variance (0.125) is lower than both individual variances (0.25 and 2.25). This demonstrates the power of diversification - by combining investments with negative covariance, we can reduce overall risk while maintaining the same expected return. The negative covariance indicates that when Tortoise performs well, Hare tends to perform poorly, and vice versa, creating a natural hedge.

This example illustrates a fundamental principle in finance and decision theory: diversification can reduce risk without sacrificing expected returns when assets are not perfectly positively correlated. The key insight is that variance depends not only on individual asset volatilities but also on their covariances, making portfolio construction a crucial consideration in risk management.

1.4 Limiting Behavior of Averages

The Tortoise and Hare example illustrates how expectations and variances behave for a single period or a fixed portfolio. However, in many real-world applications—from insurance to machine learning—we are interested in what happens when we repeat an experiment many times. Does the average outcome settle down to a predictable value?

When we observe a stochastic process repeatedly, we naturally ask: what happens to averages as we collect more data? This question lies at the heart of statistical inference and forms the theoretical foundation for learning from experience. The answer is provided by the *law of large numbers*, one of the most fundamental results in probability theory.

1.4.1 The Weak Law of Large Numbers

The weak law of large numbers, in its simplest form, states that sample averages converge in probability to the expected value. We assume that observation x_1, x_2, \dots, x_n are draws (realizations) of independent identically distributed (i.i.d.) random variables X_1, X_2, \dots, X_n with finite mean $\mu = E(X_i)$. Then the sample average $\bar{X}_n = \frac{1}{n} \sum_{i=1}^n x_i$ satisfies:

$$\lim_{n \rightarrow \infty} P(|\bar{X}_n - \mu| > \epsilon) = 0,$$

for any $\epsilon > 0$. This form of convergence, known as *convergence in probability*, means that for large enough n , the probability that the sample average deviates from the true mean by more than any fixed amount becomes arbitrarily small.

The proof of this result, when the variance $\sigma^2 = \text{Var}(X_i)$ exists and is finite, follows elegantly from Chebyshev's inequality. Since $E(\bar{X}_n) = \mu$ and $\text{Var}(\bar{X}_n) = \sigma^2/n$, we have:

$$P(|\bar{X}_n - \mu| > \epsilon) \leq \frac{\text{Var}(\bar{X}_n)}{\epsilon^2} = \frac{\sigma^2}{n\epsilon^2} \rightarrow 0$$

as $n \rightarrow \infty$. This simple argument reveals why averages become more reliable as sample sizes grow: the variance of the sample mean shrinks at rate $1/n$.

This $1/n$ scaling reappears throughout the book, from confidence intervals and A/B testing (Chapter 5) to the role of sample size in estimation risk and generalization.

1.4.2 Kolmogorov's Strong Law of Large Numbers

While the weak law establishes convergence in probability, a stronger form of convergence is possible. The *strong law of large numbers* states that sample averages converge *almost surely* (with probability one) to the expected value. This is a fundamentally stronger statement: it means that for almost every realization of the sequence, the sample average actually approaches the true mean, not merely that the probability of large deviations vanishes.

Andrey Kolmogorov formalized this result in his groundbreaking 1933 monograph *Foundations of the Theory of Probability* (Kolmogoroff 1933), establishing the conditions under which strong convergence holds. His result revolutionized probability theory by providing a rigorous measure-theoretic foundation for probabilistic reasoning.

Kolmogorov's Strong Law states that if X_1, X_2, \dots are independent random variables (not necessarily identically distributed) with $E(|X_i|) < \infty$, then:

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i \rightarrow \mu \quad \text{almost surely}$$

where $\mu = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n \mathbb{E}(X_i)$, provided this limit exists. For i.i.d. random variables with finite mean $\mathbb{E}(X_i) = \mu$, this simplifies to $\bar{X}_n \rightarrow \mu$ almost surely.

The distinction between convergence in probability and almost sure convergence is subtle but crucial. Convergence in probability allows for infinitely many large deviations, as long as they become increasingly rare. Almost sure convergence is stronger: it requires that eventually, after some finite (but random) time, all deviations remain small forever.

Formally, almost sure convergence means:

$$P\left(\lim_{n \rightarrow \infty} \bar{X}_n = \mu\right) = 1$$

This is equivalent to saying that the set of sequences for which the limit fails to equal μ has probability zero. For practical purposes, this means we can be confident that the specific sequence we observe will exhibit convergence—not merely that convergence is likely.

The independence assumption is central to Kolmogorov's results, but it can be relaxed in various ways for different applications. For stochastic processes with dependent observations, the key question becomes: how much dependence can we tolerate while still obtaining convergence of averages?

For *stationary processes*—where the joint distribution of $(X_t, X_{t+1}, \dots, X_{t+k})$ does not depend on t —a law of large numbers holds under considerably weaker conditions than independence. If the process is *ergodic* (roughly, if it eventually “forgets” its initial conditions), then time averages converge to ensemble averages:

$$\frac{1}{n} \sum_{i=1}^n X_i \rightarrow \mu \quad \text{almost surely}$$

This *ergodic theorem* extends Kolmogorov's law of large numbers to dependent sequences and has profound implications for statistical inference from time series data. It justifies the common practice of estimating population means from a single long realization of a stochastic process.

The law of large numbers is not merely a theoretical curiosity—it forms the bedrock of statistical practice. Every time we estimate a population mean from a sample, test a hypothesis, or train a machine learning model, we implicitly rely on the law of large numbers. The confidence we place in larger samples, the use of cross-validation to assess model performance, and the convergence of

stochastic gradient descent in deep learning all trace back to this fundamental result.

In the context of stochastic processes, the law of large numbers justifies estimating process parameters from a single long trajectory. When modeling financial returns, climate data, or network traffic, we typically observe one realization over time rather than multiple independent realizations. The ergodic theorem ensures that time averages from this single path converge to the true population moments, enabling inference from the data we actually have.

The Kolmogorov's formalization is the culmination of several decades of work on the foundations of probability theory. The weak law of large numbers was first proved by Jakob Bernoulli in 1713 for the special case of binomial random variables—an achievement that took him over twenty years. The result was later generalized by Poisson (1837) and Chebyshev (1867) to broader classes of random variables.

The strong law required deeper mathematical machinery. Émile Borel proved a version for Bernoulli trials in 1909, but the general result awaited the development of measure-theoretic probability. Francesco Cantelli made progress in the 1910s, but it was Kolmogorov who provided the definitive treatment in 1933, unifying diverse results under a single rigorous framework.

Kolmogorov's work transformed probability theory from a collection of special cases and heuristics into a branch of mathematics with the same rigor as analysis or algebra. His measure-theoretic foundations enabled precise statements about almost sure convergence, clarified the distinction between different modes of convergence, and opened the door to modern probability theory and stochastic processes.

The law of large numbers has profound implications for Bayesian inference and computational statistics. For applications to posterior consistency, Monte Carlo simulation, and Markov chain Monte Carlo (MCMC) methods, see Section 3.11 in Chapter 3.

1.5 Binomial, Poisson, and Normal Distributions

We now examine three fundamental probability distributions that appear throughout statistics and machine learning.

1.5.1 Bernoulli Distribution

The formal model of a coin toss was described by Bernoulli. He modeled the notion of *probability* for a coin toss, now known as the Bernoulli distribution, where $X \in \{0, 1\}$ and $P(X = 1) = p, P(X = 0) = 1 - p$. Laplace gave us the *principle of insufficient reason*: where you would list out the possibilities and then place equal probability on each of the outcomes. Essentially the discrete uniform distribution on the set of possible outcomes.

A Bernoulli trial relates to an experiment with the following conditions

1. The result of each trial is either a success or failure.
2. The probability p of a success is the same for all trials.
3. The trials are assumed to be *independent*.

The Bernoulli random variable can take on one of two possible outcomes, typically labeled as “success” and “failure.” It is named after the Swiss mathematician Jacob Bernoulli, who introduced it in the 18th century. The distribution is often denoted by $\text{Bernoulli}(p)$, where p is the probability of success.

The probability mass function (PMF) of a Bernoulli distribution is defined as follows:

$$P(X = x) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0 \end{cases}$$

The expected value (mean) of a Bernoulli distributed random variable X is given by:

$$\mathbb{E}(X) = p$$

Simply speaking, if you are to toss a coin many times, you expect p heads.

The variance of X is given by:

$$\text{Var}(X) = p(1 - p)$$

Example 1.15 (Coin Toss). The quintessential random variable is an outcome of a coin toss. The set of all possible outcomes, known as the sample space, is $S = \{H, T\}$, and $P(X = H) = P(X = T) = 1/2$. On the other hand, a single outcome can be an element of many different events. For example, there are four possible outcomes of two coin tosses, HH, TT, HT, TH, which are equally likely with probabilities $1/4$. The probability mass function over the number of heads X out of two coin tosses is

x	0	1	2
$p(x)$	1/4	1/2	1/4

Given the probability mass function we can, for example, calculate the probability of at least one head as $P(X \geq 1) = P(X = 1) + P(X = 2) = p(1) + p(2) = 3/4$.

The Bernoulli distribution serves as the foundation for more complex distributions, such as the binomial distribution (which models the number of successes in a fixed number of independent Bernoulli trials) and the geometric distribution (which models the number of trials needed to achieve the first success). A Binomial distribution arises from a sequence of Bernoulli trials, and assigns probability to X , which is the number of successes. Its probability distribution is calculated via:

$$P(X = x) = \binom{n}{x} p^x (1 - p)^{n-x}.$$

Here $\binom{n}{x}$ is the combinatorial function,

$$\binom{n}{x} = \frac{n!}{x!(n-x)!},$$

Here the combinatorial function $\binom{n}{x}$ counts the number of ways of getting x successes in n trials and $n! = n(n-1)(n-2)\dots 2 \cdot 1$ counts the number of permutations of n observations without replacement.

Plot below shows the probability mass function of a Binomial distribution with $n = 20$ and $p = 0.3$.

```
barplot(dbinom(0:20, size = 20, prob = 0.3),
        names.arg = 0:20, col = "lightblue",
        xlab = "Number of Heads", ylab = "Probability"
)
```

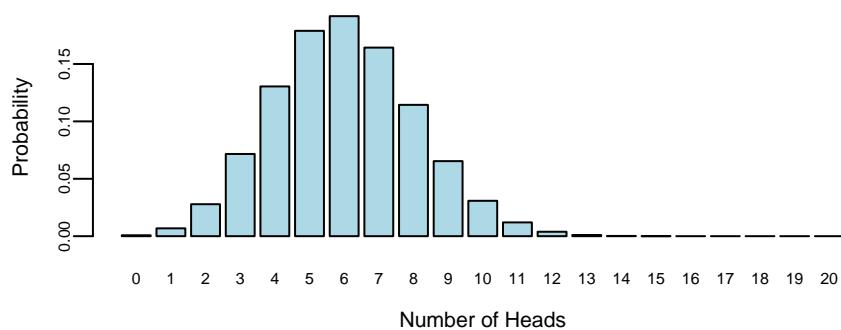


Figure 1.8: Probability Mass Function of a Binomial Distribution ($n = 20$, $p = 0.3$)

The table below shows the expected value and variance of a Binomial random variable. Those quantities can be calculated by plugging in the possible outcomes and corresponding probabilities into the definitions of expected value and variance.

Table 1.7: Mean and Variance of Binomial

Binomial Distribution	Parameters
Expected value	$\mu = E(X) = np$
Variance	$\sigma^2 = \text{Var}(X) = np(1 - p)$

For large sample sizes n , this distribution is approximately normal with mean np and variance of $np(1 - p)$.

Let X be the number of heads in three flips. Each possible outcome (“realization”) of X is an *event*. Now consider the event of getting only two heads

$$\{X = 2\} = \{HHT, HTH, THH\},$$

The probability distribution of X is Binomial with parameters $n = 3, p = 1/2$, where n denotes the sample size (a.k.a. number of trials) and p is the probability of heads; we have a fair coin. The notation is $X \sim \text{Bin}(n = 3, p = \frac{1}{2})$ where the sign \sim is read as *distributed as*.

Table 1.8: Outcomes of three coin flips

Result	X	$P(X = x)$
HHH	3	p^3
HHT	2	$p^2(1 - p)$
HTH	2	$p^2(1 - p)$
THH	2	$(1 - p)p^2$
HTT	1	$p(1 - p)^2$
THT	1	$p(1 - p)^2$
TTH	1	$(1 - p)^2p$
TTT	0	$(1 - p)^3$

1.5.2 Poisson Distribution

The Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant mean rate and independently of the time since the last event. You can think of Poisson distribution as a limiting case of the Binomial distribution when the number of trials n

is large and the probability of success p is small, such that $np = \lambda$ remains constant. Think of a soccer game where the goal is the “successful” event of interest. The soccer team does not have a predefined number of attempts to score a goal. Rather they continuously try to score a goal until they do. The number of goals scored in a game is the number of events occurring in a fixed interval of time. The mean number of goals scored in a game is the mean rate of events occurring in a fixed interval of time.

There are many examples of Poisson distributed random variables. For example, the number of phone calls received by a call center per hour, the number of emails received per day, the number of customers arriving at a store per hour, the number of defects in a manufactured product, the number of accidents on a highway per month.

A random variable X follows a Poisson distribution with parameter $\lambda > 0$ if its probability mass function is given by:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

for $k = 0, 1, 2, 3, \dots$, where λ is both the mean and variance of the distribution.

Plot below shows the probability mass function of a Poisson distribution with $\lambda = 1, 5, 10$.

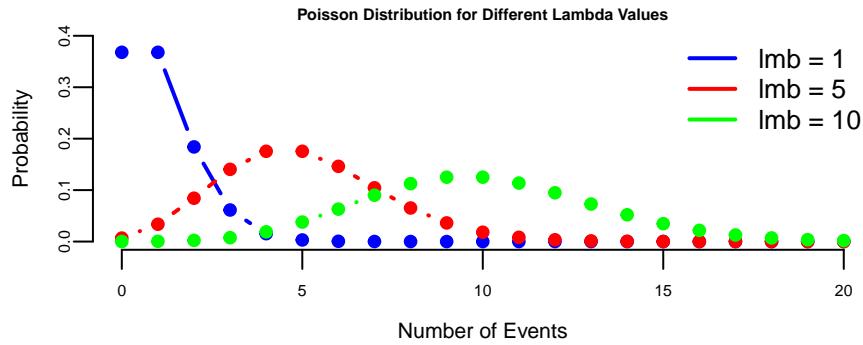


Figure 1.9: Probability Mass Function of a Poisson Distribution

We write $X \sim \text{Poisson}(\lambda)$ to denote that X follows a Poisson distribution with parameter λ .

Table 1.9: Mean and Variance of Poisson

Poisson Distribution	Parameters
Expected value	$\mu = E(X) = \lambda$

Poisson Distribution	Parameters
Variance	$\sigma^2 = \text{Var}(X) = \lambda$

Example 1.16 (Customer Arrivals). Suppose customers arrive at a coffee shop at an average rate of 3 customers per hour. What is the probability that exactly 5 customers will arrive in the next hour?

Using the Poisson distribution with $\lambda = 3$:

$$P(X = 5) = \frac{3^5 e^{-3}}{5!} = \frac{243 \times e^{-3}}{120} \approx 0.101$$

So there is approximately a 10.1% chance that exactly 5 customers will arrive in the next hour.

The Poisson distribution can be derived as a limiting case of the binomial distribution when n is large and p is small, such that $np = \lambda$ remains constant. This connection makes the Poisson distribution particularly useful for modeling rare events in large populations.

1.5.3 Normal Distribution

The Normal distribution is a continuous probability distribution that is widely used in statistics and probability theory. It is also known as the Gaussian distribution or the bell curve. The Normal distribution is characterized by its symmetric bell-shaped curve and is defined by two parameters: the mean (μ) and the variance (σ^2).

The probability density function (PDF) of the Normal distribution is given by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

for $-\infty < x < \infty$, where μ is the mean and σ is the standard deviation.

The Normal distribution is often used to model real-world phenomena such as measurement errors, heights, weights, and scores on standardized tests. It is also used in hypothesis testing and confidence interval construction.

Example 1.17 (Heights of Adults). The heights of adult males in a certain population are normally distributed with a mean of 70 inches and a standard deviation of 3 inches.

- a) What is the probability that a randomly selected male is between 67 and 73 inches tall?

Using the Normal distribution with $\mu = 70$ and $\sigma = 3$:

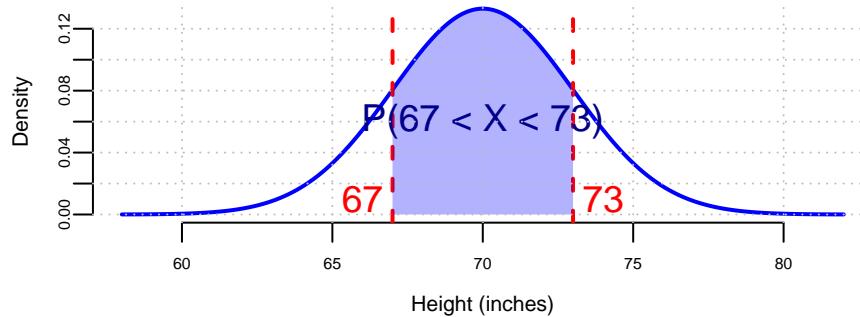


Figure 1.10: Normal Distribution ($\mu = 70$, $\sigma = 3$) with Highlighted Area

The bell curve in Figure 1.10 shows the probability density function of a random variable $X \sim N(70, 3^2)$ that follows a Normal distribution with mean 70 and standard deviation 3. As we can see, the Normal distribution is symmetric around the mean, and the mean, median, and mode are all equal. The probability of a randomly selected male being between 67 and 73 inches tall is the area under the curve between 67 and 73 inches and is approximately 0.6827.

Now let's calculate this probability using R:

```
prob_between <- pnorm(73, mean = 70, sd = 3) - pnorm(67, mean =
  ↪ 70, sd = 3)
cat("P(67 < X < 73) =", round(prob_between, 4), "\n")
## P(67 < X < 73) = 0.68
```

The calculation shows that $P(67 < X < 73) = P(X \leq 73) - P(X \leq 67) \approx 0.6827$.

This result makes sense because 67 and 73 inches are exactly one standard deviation below and above the mean (70 ± 3), respectively. According to the empirical rule (68-95-99.7 rule), approximately 68% of values in a normal distribution fall within one standard deviation of the mean.

So approximately 68.27% of adult males are between 67 and 73 inches tall.

b) What height corresponds to the 95th percentile?

We need to find x such that $P(X \leq x) = 0.95$. From standard normal tables, $\Phi^{-1}(0.95) \approx 1.645$.

```
x <- qnorm(0.95, mean = 70, sd = 3)
cat("Height corresponding to the 95th percentile:", round(x, 2),
    "inches\n")
## Height corresponding to the 95th percentile: 75 inches
```

Therefore: $x = \mu + \sigma \cdot 1.645 = 70 + 3 \cdot 1.645 = 74.935$ inches.

1.6 Conditional, Marginal and Joint Distributions

Suppose that we have two random variables X and Y , which can be related to each other. Knowing X would change your belief about Y . For example, as a first pass, psychologists who study the phenomenon of happiness can be interested in understanding its relation to income level. Now we need a single probability mass function (a.k.a. probabilistic model) that describes all possible values of those two variables. Joint distributions do exactly that.

Formally, the *joint distribution* of two variables X and Y is a function given by

$$P(x, y) = P(X = x, Y = y).$$

This maps all combinations of possible values of these two variables to a probability on the interval $[0,1]$.

The *conditional probability* is a measure of the probability of a random variable X , given that the value of another random variable was observed $Y = y$.

$$P(x | y) = P(X = x | Y = y).$$

The *marginal probability* of a subset of a collection of random variables is the probability distribution of the variables contained in the subset without reference to the values of the other variables. Say we have two random variables X and Y , the marginal probability $P(X)$ is the probability distribution of X when the values of Y are not taken into consideration. This can be calculated by summing the joint probability distribution over all values of Y . The converse is also true: the marginal distribution can be obtained for Y by summing over the separate values of X .

Marginal probability is different from conditional probability. Marginal probability is the probability of a single event occurring, independent of other events. A conditional probability, on the other hand, is the probability that an event occurs given that another specific event has already occurred.

Example 1.18 (Salary-Happiness). Let's look at an example. Suppose that to model the relationship between two quantities, salary Y and happiness X . After running a survey, we summarize our results using the joint distribution, that is described by the following “happiness index” table as a function of salary.

Table 1.10: Results of the Gallup survey. Rows are Salary (Y) and columns are happiness (X)

	$X = 0$ (low)	$X = 1$ (medium)	$X = 2$ (high)
$Y = \text{low}$ (0)	0.03	0.13	0.14
$Y = \text{medium}$ (1)	0.12	0.11	0.01
$Y = \text{high}$ (2)	0.07	0.01	0.09
$Y = \text{very high}$ (3)	0.02	0.13	0.14

Each cell of the table is the joint probability, e.g. 14% of people have very high income level and are very happy. Those joint probabilities are calculated by simple counting and calculating the proportions.

Now, if we want to answer the question what is the percent of high earners in the population. For that we need to calculate what is called a *marginal probability* $P(y = 2)$. We can calculate the proportion of high earners $P(y = 2)$ by summing up the entries in the third row of the table, which is 0.17 in our case.

```
0.07 + 0.01 + 0.09
## 0.17
```

Formally marginal probability over y is calculated by summing the joint probability over the other variable, x ,

$$p(y) = \sum_{x \in S} p(x, y)$$

Where S is the set of all possible values of the random variable X .

Another question of interest is whether happiness depends on income level. To answer those types of questions, we need to introduce an important concept, which is the *conditional probability* of X given that the value of variable Y is known. This is denoted by $P(X = x | Y = y)$ or simply $p(x | y)$, where $|$ reads as “given” or “conditional upon”.

The conditional probability $p(x | y)$ also has interpretation as updating your probability over X after you have learned the new information about Y . In this sense, probability is also the language of how you change opinions in light

of new evidence. The proportion of happy people among high earners is given by the conditional probability $P(X = 2 | Y = 2)$ and can be calculated by dividing the proportion of those who are high earners and highly happy by the proportion of high earners

$$P(X = 2 | Y = 2) = \frac{P(X = 2, Y = 2)}{P(Y = 2)} = \frac{0.09}{0.17} = 0.5294118.$$

Now, if we compare it with the proportion of highly happy people $P(X = 2) = 0.38$, we see that on average you are more likely to be happy given your income is high.

1.7 Independence

Historically, the concept of independence in experiments and random variables has been a defining mathematical characteristic that has uniquely shaped the theory of probability. This concept has been instrumental in distinguishing the theory of probability from other mathematical theories.

Using the notion of conditional probability, we can define independence of two variables. Two random variables X and Y are said to be *independent* if

$$P(Y = y | X = x) = P(Y = y),$$

for all possible x and y values. That is, learning information $X = x$ doesn't affect our probabilistic assessment of Y for any value y . In the case of independence, $p(x | y) = p(x)$ and $p(y | x) = p(y)$.

Conditional probabilities are counter-intuitive. For example, one of the most important properties is typically $p(x | y) \neq p(y | x)$. Confusing these two—specifically equating $P(A|B)$ with $P(B|A)$ —is a common error known as the *Prosecutor's Fallacy*.

We just derived an important relation that allows us to calculate conditional probability $p(x | y)$ when we know joint probability $p(x, y)$ and marginal probability $p(y)$. The total probability or evidence can be calculated as usual, via $p(y) = \sum_x p(x, y)$.

We will see that independence will lead to a different conclusion than the Bayes conditional probability decomposition: specifically, independence yields $p(x, y) = p(x)p(y)$ whereas Bayes implies $p(x, y) = p(x)p(y | x)$.

1.8 Further Notes: Dutch Book Arguments

If probabilities are degrees of belief and subjective, where do they come from and what rules must they satisfy? These questions were answered to varying degrees by Ramsey, de Finetti, and Savage. Ramsey and de Finetti, working independently and at roughly the same time, developed the first primitive theories of subjective probability and expected utility, and Savage placed the theories on a more rigorous footing, combining the insights of Ramsey with the expected utility theory of von Neumann and Morgenstern.

The starting point for Ramsey's and de Finetti's theories is the measurement of one's subjective probabilities using betting odds, which have been used for centuries to gauge the uncertainty over an event. As noted by de Finetti, “*It is a question of simply making mathematically precise the trivial and obvious idea that the degree of probability attributed by an individual to a given event is revealed by the conditions under which he would be disposed to bet on that event*” (p. 101). Notice the difference between the frequentist and Bayesian approach. Instead of defining the probabilities via an infinite repeated experiment, the Bayesian approach elicits probabilities from an individual's observed behavior.

Formally, for any event A , the identity

$$P(A) = \frac{1}{1 + \text{odds}(A)} \text{ or } \text{odds}(A) = \frac{1 - P(A)}{P(A)},$$

where \bar{A} is the complement of A , links odds and probabilities. Throughout, we use P as a generic term to denote probabilities, when there is no specific reference to an underlying distribution or density. If a horse in a race has odds of 2, commonly expressed as 2:1 (read two to one), then the probability the horse wins is $1/3$. The basic idea of using betting odds to elicit probabilities is simple and intuitive: ask an individual to place odds over various mutually exclusive events, and use these odds to calculate the probabilities. Odds are *fair* if lower odds would induce a person to take the bet and higher odds would induce the person to take the other side of the bet.

In constructing a collection of betting odds over various events, de Finetti and Ramsey argued that not all odds are rational (i.e., consistent or coherent). For example, the sum of the probability of each horse winning a race cannot be greater than one. If a person has inconsistent beliefs, then he “*could have a book made against him by a cunning bettor and would then stand to lose in any event*” (Ramsey (1931), p. 22). This situation is called a Dutch book arbitrage, and a rational theory of probability should rule out such inconsistencies. By avoiding Dutch books, Ramsey and de Finetti showed that the degrees of beliefs elicited from coherent odds satisfy the standard axioms of probability theory, such as the restriction that probabilities are between zero

and one, finite additivity, and the laws of conditional probability. The converse also holds: probabilities satisfying the standard axioms generate odds excluding Dutch-book arbitrages. Absence of arbitrage is natural in finance and economics and is a primary assumption for many foundational results in asset pricing. In fact, the derivations given below have a similar flavor to those used to prove the existence of a state price density assuming discrete states.

Dutch-book arguments are simple to explain. To start, they require an individual to post odds over events. A bettor or bookie can then post stakes or make bets at those odds with a given payoff, S . The choice of the stakes is up to the bettor. A Dutch book occurs when a cunning bettor makes money for sure by placing carefully chosen stakes at the given odds. Alternatively, one can view the odds as prices of lottery tickets that pay off \$1 when the event occurs, and the stakes as the number of tickets bought. Thus, probabilities are essentially lottery ticket prices. In fact, de Finetti used the notation ‘Pr’ to refer to both prices and probabilities.

To derive the rules, consider the first axiom of probability: for any event A , $0 \leq P(A) \leq 1$. Suppose that the odds imply probabilities $P(A)$ for A occurring and $P(\bar{A})$ for other outcomes, with associated payoffs of S_A and $S_{\bar{A}}$. Then, having bet S_A and $S_{\bar{A}}$, the gains if A or \bar{A} occur, G_A and $G_{\bar{A}}$, respectively, are

$$\begin{aligned} G(A) &= S_A - P(A)S_A - P(\bar{A})S_{\bar{A}} \\ G(\bar{A}) &= S_{\bar{A}} - P(A)S_A - P(\bar{A})S_{\bar{A}}. \end{aligned}$$

To see this, note that the bettor receives S_A and pays $P(A)S_A$ for a bet on event A . The bookie can always choose to place a zero stake on \bar{A} occurring, which implies that $G(A) = S_A - P(A)S_A$ and $G(\bar{A}) = -P(A)S_A$. Coherence or the absence of arbitrage implies that you cannot gain or lose in both states, thus $G(A)G(\bar{A}) \leq 0$. Substituting, $(1 - P(A))P(A) \geq 0$ or $0 \leq P(A) \leq 1$, which is the first axiom of probability. The second axiom, that the set of all possible outcomes has probability 1, is similarly straightforward to show.

The third axiom is that probabilities add, that is, for two disjoint events A_1 and A_2 , $P(A) = P(A_1 \text{ or } A_2) = P(A_1) + P(A_2)$. Assuming stakes sizes of S_A , S_{A_1} , and S_{A_2} (and zero stakes on their complements) there are three possible outcomes. If neither A_1 nor A_2 occur, the gain is

$$G(\bar{A}) = -P(A)S_A - P(A_1)S_{A_1} - P(A_2)S_{A_2}.$$

If A_1 occurs, A also occurs, and the gain is

$$G(A_1) = (1 - P(A))S_A + (1 - P(A_1))S_{A_1} - P(A_2)S_{A_2},$$

and finally if A_2 occurs, A also occurs, and

$$G(A_2) = (1 - P(A))S_A - P(A_1)S_{A_1} + (1 - P(A_2))S_{A_2}.$$

Arranging these into a matrix equation, $G = PS$:

$$\begin{pmatrix} G(\bar{A}) \\ G(A_1) \\ G(A_2) \end{pmatrix} = \begin{pmatrix} -P(A) & -P(A_1) & -P(A_2) \\ 1 - P(A) & 1 - P(A_1) & -P(A_2) \\ 1 - P(A) & -P(A_1) & 1 - P(A_2) \end{pmatrix} \begin{pmatrix} S_A \\ S_{A_1} \\ S_{A_2} \end{pmatrix}.$$

The absence of a Dutch book arbitrage implies that there is no set of stakes, S_A , S_{A_1} , and S_{A_2} , such that the winnings in all three events are positive. If the matrix P is invertible, it is possible to find stakes with positive gains. To rule this out, the determinant of P must be zero, which implies that $0 = -P(A) + P(A_1) + P(A_2)$, or $P(A) = P(A_1) + P(A_2)$.

The fourth axiom is conditional probability. Consider an event B , with $P(B) > 0$, an event A that occurs conditional on B , and the event that both A and B occur. The probabilities or prices of these bets are $P(B)$, $P(A | B)$, and $P(A \text{ and } B)$. Consider bets with stakes S_B , $S_{A|B}$ and $S_{A \text{ and } B}$, with the understanding that if B does not occur, the conditional bet on A is canceled. The payoffs to the events that B does not occur, B occurs but not A , and A and B occur, are

$$\begin{pmatrix} G(\bar{B}) \\ G(\bar{A} \text{ and } B) \\ G(A \text{ and } B) \end{pmatrix} = \begin{pmatrix} -P(B) & -P(A \text{ and } B) & 0 \\ 1 - P(B) & -P(A \text{ and } B) & -P(A | B) \\ 1 - P(B) & 1 - P(A \text{ and } B) & 1 - P(A | B) \end{pmatrix} \begin{pmatrix} S_B \\ S_{A \text{ and } B} \\ S_{A|B} \end{pmatrix}.$$

Similar arguments imply the determinant must be zero, which implies that

$$P(A | B) = \frac{P(A \text{ and } B)}{P(B)},$$

which is the law of conditional probability, given $P(B) > 0$, of course, otherwise the conditional probability is not defined, and the P matrix has determinant 0.

2

Bayes Rule

When the facts change, I change my mind. What do you do, sir? John Maynard Keynes

One of the key questions in the theory of learning is: *How do you update your beliefs in the presence of new information?* Bayes rule provides the answer. Conditional probability can be interpreted as updating your probability of event A after you have learned the new information that B has occurred. In this sense probability is also the language of how you'll change opinions in the light of new evidence.

Probability rules allow us to change our mind if the facts change. For example, suppose that we have evidence $E = \{E_1, E_2\}$ consists of two pieces of information and that we are interested in identifying a **cause** C . Bayes rule simply lets you calculate this conditional probability $P(C | E_1, E_2)$ in a sequential fashion. First, conditioning on the information contained in E_1 , lets us calculate

$$P(C|E_1) = \frac{P(E_1 | C)P(C)}{P(E_1)}$$

Then, using the posterior probability $P(C|E_1)$ as the “new” prior for the next piece of information E_2 lets us find

$$P(C|E_1, E_2) = \frac{P(E_2 | E_1, C)P(C | E_1)}{P(E_2 | E_1)}$$

Hence, we see that we need assessments of the two conditional probabilities $P(E_1 | C)$ and $P(E_2 | E_1, C)$. In many situations, the latter will be simply $P(E_2 | C)$ and not involve E_1 . The events (E_1, E_2) will be said to be conditionally independent given C .

This concept generalizes to a sequence of events where $E = \{E_1, \dots, E_n\}$. When learning from data we will use this property all the time. An illustrative example will be the Black Swan problem which we discuss later.

Bayes’ rule is a fundamental concept in probability theory and statistics. It describes how to update our *beliefs* about an event based on *new evidence*. We start with an initial belief about the probability of an event (called the *prior probability*). We then observe some conditional information (e.g. evidence). We

use Bayes' rule to update our initial belief based on the evidence, resulting in a new belief called the *posterior probability*. Remember, the formula is

Bayes' rule

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

where:

- $P(A | B)$ is the **posterior probability** of event A occurring given that B is known to happen for sure. This is the probability we're trying to find.
- $P(B | A)$ is the **likelihood** of observing event B if event A has occurred.
- $P(A)$ is the **prior probability** of event A occurring. This is our initial belief about the probability of A before we see any evidence.
- $P(B)$ is the **marginal probability** of observing event B . This is the probability of observing B regardless of whether A occurs.

The ability to use Bayes rule sequentially is key in many applications, when we need to update our beliefs in the presence of new information. For example, Bayesian learning was used by mathematician Alan Turing in England at Bletchley Park to break the German Enigma code - a development that helped the Allies win the Second World War (Simpson 2010). Turing called his algorithm Banburismus, it is a process he invented which used sequential conditional probability to infer information about the likely settings of the Enigma machine.

Dennis Lindley argued that we should all be trained in Bayes rule and conditional probability can be simply viewed as disciplined probability accounting. Akin to how market odds change as evidence changes. However, human intuition is rarely naturally calibrated for Bayesian reasoning; it is a skill that must be learned, much like literacy.

2.1 Law of Total Probability

The Law of Total Probability is a fundamental rule relating marginal probabilities to conditional probabilities. It's particularly useful when you're dealing with a set of mutually exclusive and collectively exhaustive events.

Suppose you have a set of events B_1, B_2, \dots, B_n that are mutually exclusive (i.e., no two events can occur at the same time) and collectively exhaustive (i.e.,

at least one of the events must occur). The Law of Total Probability states that for any other event A , the probability of A occurring can be calculated as the sum of the probabilities of A occurring given each B_i , multiplied by the probability of each B_i occurring.

Mathematically, it is expressed as:

$$P(A) = \sum_{i=1}^n P(A | B_i)P(B_i)$$

Example 2.1 (Total Probability). Let's consider a simple example to illustrate this. Suppose you have two bags of balls. Bag 1 contains 3 red and 7 blue balls, while Bag 2 contains 6 red and 4 blue balls. You randomly choose one of the bags and then randomly draw a ball from that bag. What is the probability of drawing a red ball?

Here, the events B_1 and B_2 can be choosing Bag 1 and Bag 2, respectively. You want to find drawing a red ball (event A).

Applying the law:

- $P(A | B_1)$ is the probability of drawing a red ball from Bag 1, which is $\frac{3}{10}$.
- $P(A | B_2)$ is the probability of drawing a red ball from Bag 2, which is $\frac{6}{10}$.
- Assume the probability of choosing either bag is equal, so $P(B_1) = P(B_2) = \frac{1}{2}$.

Using the Law of Total Probability:

$$P(A) = P(A | B_1) \times P(B_1) + P(A | B_2) \times P(B_2) = \frac{3}{10} \times \frac{1}{2} + \frac{6}{10} \times \frac{1}{2} = \frac{9}{20}$$

So, the probability of drawing a red ball in this scenario is $\frac{9}{20}$.

This law is particularly useful in complex probability problems where direct calculation of probability is difficult. By breaking down the problem into conditional probabilities based on relevant events, it simplifies the calculation and helps to derive a solution.

2.2 Intuition and Simple Examples

Example 2.2 (Intuition). Our intuition is not well trained to make use of Bayes rule. Suppose I tell you that Steve was selected at random from a

representative sample, and that he is 6 feet 2 inches tall and an excellent basketball player. He goes to the gym every day and practices hard playing basketball. Do you think Steve is a custodian at a factory or an NBA player? Most people assume Steve is an NBA player which is wrong. The ratio of NBA players to custodians is very small, probabilistically Steve is more likely to be a custodian. Let's look at it graphically. The key is to provide the right conditioning and to consider the prior probability! Even though the ratio of people who practice basketball hard is much higher among NBA players (it is 1) when compared to custodians, the larger number of the population means we still have more custodians in the US than NBA players.

$$\begin{aligned} P(\text{Practice hard} \mid \text{Play in NBA}) &\approx 1 \\ P(\text{Play in NBA} \mid \text{Practice hard}) &\approx 0. \end{aligned}$$

Even though you practice hard, the odds of playing in the NBA are low (1000 players out of 7 billion). But given you're in the NBA, you no doubt practice very hard. To understand this further, let's look at the conditional probability implication and apply Bayes rule

$$P(\text{Play in NBA} \mid \text{Practice hard}) = \frac{P(\text{Practice hard} \mid \text{Play in NBA})}{P(\text{Practice hard})} P(\text{Play in NBA}).$$

This is written in the form

$$\text{Posterior} = \frac{\text{Likelihood}}{\text{Marginal}} \times \text{Prior} = \text{Bayes Factor} \times \text{Prior}.$$

The Likelihood/Marginal ratio is called the Bayes Factor. As we will see in the text, one of the key advantages over classical is the ability to sequentially update our beliefs as new evidence appears. It allows for disciplined probability accounting in “real-time”. With the advent of prediction markets and data science, it has become increasingly important to be able to update our beliefs as new evidence appears.

The initial (a.k.a. prior) probability $P(\text{Play in NBA}) = 450/(8 \cdot 10^9) = 5.625 \times 10^{-8}$. This assumes a global population of around 8 billion, making the conditional (or, so called, posterior) probability also very small.

$$P(\text{Play in NBA} \mid \text{Practice hard}) \approx 0,$$

$P(\text{practice hard})$ is not that small and $P(\text{practice hard} \mid \text{play in NBA}) = 1$. Hence, when one ‘reverses the conditioning’ one gets a very small probability. This makes sense!

The Steve example illustrates how our intuition fails us, but let's consider an even more striking case that demonstrates the power of Bayes rule with

extreme probabilities. Consider the question: what is the probability that a randomly selected 7-foot-tall American male plays in the NBA?

Most people's intuition suggests this probability should be quite high - after all, being exceptionally tall seems like the primary qualification for professional basketball. However, Bayes rule reveals a more nuanced picture that depends critically on the base rates involved.

To calculate $P(\text{NBA player} \mid 7 \text{ feet tall})$ using Bayes rule, we need to carefully estimate each component:

$$P(\text{NBA} \mid 7\text{ft}) = \frac{P(7\text{ft} \mid \text{NBA}) \times P(\text{NBA})}{P(7\text{ft})}$$

The prior probability $P(\text{NBA})$ represents the baseline chance of being an NBA player. With approximately 450 active players drawn from roughly 40 million American males of playing age, this gives us $P(\text{NBA}) \approx 1.1 \times 10^{-5}$ - an extraordinarily small number.

The likelihood $P(7\text{ft} \mid \text{NBA})$ asks what fraction of NBA players are 7 feet or taller. Historically, this has been around 17% of the league, so $P(7\text{ft} \mid \text{NBA}) \approx 0.17$.

The marginal probability $P(7\text{ft})$ requires us to estimate how rare 7-foot-tall men are in the general population. Male height follows approximately a normal distribution with mean 69 inches and standard deviation 3 inches. At 84 inches (7 feet), we're looking at a z-score of 5.0, which corresponds to roughly 1 in 3.5 million men, giving us $P(7\text{ft}) \approx 2.87 \times 10^{-7}$.

Applying Bayes rule:

$$P(\text{NBA} \mid 7\text{ft}) = \frac{0.17 \times 1.1 \times 10^{-5}}{2.87 \times 10^{-7}} \approx 0.065$$

This yields approximately 6.5% - a dramatic increase from the baseline probability of 0.001%, yet still surprisingly low given our intuitions.

Let's also consider a direct count-based calculation. As of September 2025, there are 39 players in the NBA who are 7 feet tall or taller, out of a total of 450 NBA players. This means the probability that a randomly selected NBA player is at least 7 feet tall is:

$$P(7\text{ft} \mid \text{NBA}) = \frac{39}{450} = 0.0867$$

This empirical estimate is slightly different from the earlier historical average, but it still highlights the rarity of extreme height even among elite basketball players.

Regardless of the exact calculation, this example powerfully demonstrates how Bayes rule forces us to account for base rates. Even when height provides enormous predictive value for NBA success (the likelihood ratio is massive), the extreme rarity of both 7-foot-tall individuals and NBA players means that most 7-footers will not be professional basketball players. This counterintuitive result exemplifies why disciplined probabilistic reasoning through Bayes rule is essential for making accurate inferences in the presence of rare events.

Example 2.3 (Craps). Craps is a fast-moving dice game with a complex betting layout. It's highly volatile, but eventually your bankroll will drift towards zero. Let's look at the pass line bet. The expectation $E(X)$ governs the long run. When 7 or 11 comes up, you win. When 2, 3 or 12 comes up, this is known as "craps", you lose. When 4, 5, 6, 8, 9 or 10 comes up, this number is called the "point", the bettor continues to roll until a 7 (you lose) or the point comes up (you win).

We need to know the probability of winning. The pay-out, probability and expectation for a \$1 bet

Win	Prob
1	0.4929
-1	0.5071

This leads to an edge in favor of the house as

$$E(X) = 1 \cdot 0.4929 + (-1) \cdot 0.5071 = -0.014$$

The house has a 1.4% edge.

To calculate the probability of winning: $P(\text{Win})$ let's use the law of total probability

$$P(\text{Win}) = \sum_{\text{Point}} P(\text{Win} | \text{Point})P(\text{Point})$$

The set of $P(\text{Point})$ are given by

Value	Probability	Percentage	Value	Probability	Percentage
2	1/36	2.78%	8	5/36	13.9%
3	2/36	5.56%	9	4/36	11.1%
4	3/36	8.33%	10	3/36	8.33%
5	4/36	11.1%	11	2/36	5.56%
6	5/36	13.9%	12	1/36	2.78%
7	6/36	16.7%			

The conditional probabilities $P(\text{Win} \mid \text{Point})$ are harder to calculate

$$P(\text{Win} \mid 7 \text{ or } 11) = 1 \text{ and } P(\text{Win} \mid 2, 3 \text{ or } 12) = 0$$

We still have to work out all the probabilities of winning given the point.
Suppose the point is 4

$$P(\text{Win} \mid 4) = P(4 \text{ before } 7) = \frac{P(4)}{P(7) + P(4)} = \frac{3}{9} = \frac{1}{3}$$

There are 6 ways of getting a 7, 3 ways of getting a 4 for a total of 9 possibilities.
Now do all of them and sum them up. You get

$$P(\text{Win}) = 0.4929$$

Hence, the casino has a slight edge of 1.4% in the long run!

Example 2.4 (Coin Jar). Large jar containing 1024 fair coins and one two-headed coin. You pick one at random and flip it 10 times and get all heads. What's the probability that the coin is the two-headed coin? The probability of initially picking the two headed coin is $1/1025$. There is a $1/1024$ chance of getting 10 heads in a row from a fair coin. Therefore, it's a 50/50 bet.

Let's do the formal Bayes rule math. Let E be the event that you get 10 Heads in a row, then

$$P(\text{two headed} \mid E) = \frac{P(E \mid \text{two headed}) P(\text{two headed})}{P(E \mid \text{fair}) P(\text{fair}) + P(E \mid \text{two headed}) P(\text{two headed})}$$

Therefore, the posterior probability

$$P(\text{two headed} \mid E) = \frac{1 \times \frac{1}{1025}}{\frac{1}{1024} \times \frac{1024}{1025} + 1 \times \frac{1}{1025}} = 0.50$$

What's the probability that the next toss is a head? Using the law of total probability gives

$$\begin{aligned} P(H) &= P(H \mid \text{two headed}) P(\text{two headed} \mid E) + P(H \mid \text{fair}) P(\text{fair} \mid E) \\ &= 1 \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = \frac{3}{4} \end{aligned}$$

Example 2.5 (Monty Hall Problem). Another example of a situation when calculating probabilities is counterintuitive. The Monty Hall problem was named after the host of the long-running TV show Let's Make a Deal. The original solution was proposed by Marilyn vos Savant, who had a column with the correct answer that many Mathematicians thought was wrong!

The game set-up is as follows. A contestant is given the choice of 3 doors. There is a prize (a car, say) behind one of the doors and something worthless behind the other two doors: two goats. The game is as follows:

1. You pick a door.
2. Monty then opens one of the other two doors, revealing a goat.
He can't open your door or show you a car
3. You have the choice of switching doors.

The question is, is it advantageous to switch? The answer is yes. The probability of winning if you switch is $2/3$ and if you don't switch is $1/3$.

Conditional probabilities allow us to answer this question. Assume you pick door 1. Let event A denote that the car is behind Door 2, and let event B denote that the host opened Door 3, revealing a goat. We want to calculate $P(A | B)$ —the probability the car is behind the door you should switch to. The prior probability that the car is behind Door 2 is $P(A) = 1/3$. If the car is behind Door 2, the host must open Door 3 (he cannot open your Door 1 or show the car behind Door 2), so $P(B | A) = 1$. Bayes rule gives us

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)} = \frac{1/3}{1/2} = \frac{2}{3}.$$

The overall probability of the host opening Door 3

$$P(B) = (1/3 \times 1/2) + (1/3 \times 1) = 1/6 + 1/3 = 1/2.$$

The posterior probability that the car is behind Door 2 after the host opens Door 3 is $2/3$. It is to your advantage to switch doors.

2.3 Real World Bayes

The true power of Bayes rule emerges in high-stakes decisions. The following cases—search-and-rescue operations, wartime survival, and legal proceedings—share a common insight: updating beliefs systematically while accounting for base rates and the direction of conditioning.

Example 2.6 (USS Scorpion sank 5 June, 1968 in the middle of the Atlantic.). Experts placed bets on each casualty and how each would affect the sinking. Undersea soundings gave a prior on location. Bayes rule: L is location and S is scenario

$$P(L | S) = \frac{P(S | L)P(L)}{P(S)}$$

The Navy spent 5 months looking and found nothing. Built a probability map: within 5 days, the submarine was found within 220 yards of the most likely probability!

A similar story happened during the search of an Air France plane that flew from Rio to Paris.

Example 2.7 (Wald and Airplane Safety). Many lives were saved by analysis of conditional probabilities performed by Abraham Wald during the Second World War. He was analyzing damages on the US planes that came back from bombing missions in Germany. Somebody suggested to analyze the distribution of the hits over different parts of the plane. The idea was to find a pattern in the damages and design a reinforcement strategy.

After examining hundreds of damaged airplanes, researchers came up with the following table

Location	Number of Planes
Engine	53
Cockpit	65
Fuel system	96
Wings, fuselage, etc.	434

We can convert those counts to probabilities

Location	Number of Planes
Engine	0.08
Cockpit	0.1
Fuel system	0.15
Wings, fuselage, etc.	0.67

We can conclude that the most likely area to be damaged on the returned planes was the wings and fuselage.

$$P(\text{hit on wings or fuselage} \mid \text{returns safely}) = 0.67$$

Wald realized that analyzing damages only on survived planes is not the right approach. Instead, he suggested that it is essential to calculate the inverse probability

$$P(\text{returns safely} \mid \text{hit on wings or fuselage}) = ?$$

To calculate that, he interviewed many engineers and pilots, he performed a lot of field experiments. He analyzed likely attack angles. He studied the properties of a shrapnel cloud from a flak gun. He suggested to the army that they fire thousands of dummy bullets at a plane sitting on the tarmac. Wald constructed a ‘probability model’ carefully to reconstruct an estimate for the joint probabilities. The table below shows the results.

	Hit	Returned	Shot Down
Engine	53	57	
Cockpit	65	46	
Fuel system	96	16	
Wings, fuselage, etc.	434	33	

Which allows us to estimate joint probabilities, for example

$$P(\text{outcome} = \text{returned safely, hit} = \text{engine}) = 53/800 = 0.066$$

We also can calculate the conditional probabilities now

$$P(\text{outcome} = \text{returned safely} \mid \text{hit} = \text{wings or fuselage}) = \frac{434}{434 + 33} = 0.9293362.$$

Should we reinforce wings or fuselage? Which part of the airplane needs to be reinforced?

$$P(\text{outcome} = \text{returned safely} \mid \text{hit} = \text{engine}) = \frac{53}{53 + 57} = 0.48$$

Bayes Rule in Odds Form

A convenient way to perform Bayes rule calculations is using the “odds” form.

$$\underbrace{\frac{O(H \mid E)}{\text{Posterior Odds}}}_{\text{Bayes Factor}} = \underbrace{\frac{P(E \mid H)}{P(E \mid \neg H)}}_{\text{Bayes Factor}} \times \underbrace{\frac{O(H)}{\text{Prior Odds}}}_{\text{Prior Odds}}$$

where the odds of an event H are defined as $O(H) = \frac{P(H)}{P(\neg H)}$. This separates the contribution of the evidence (Bayes Factor) from the prior belief.

Example 2.8 (The Prosecutor’s Fallacy). The Prosecutor’s Fallacy is a logical error that occurs when a prosecutor presents evidence or statistical data in a way that suggests a defendant’s guilt, even though the evidence is not as conclusive as it may seem. This fallacy arises from confounding conditional probabilities, specifically equating the probability of evidence given guilt $P(E|G)$ with the probability of guilt given evidence $P(G|E)$.

$$P(E \mid G) \neq P(G \mid E)$$

A classic example involves DNA evidence. Suppose you’re serving on a jury in a city with a population of 10 million. A defendant is accused of a crime based on a DNA match found at the scene. A forensic scientist testifies that

the probability of an innocent person's DNA matching the sample is one in a million ($P(E|\bar{G}) = 10^{-6}$).

The prosecutor argues that this means there is only a one in a million chance the defendant is innocent. This is the fallacy. You are charged with assessing $P(G | E)$ - the probability of guilt given the match.

Let's apply Bayes' rule. - $P(G) \approx 1/10^7$ (Prior probability, assuming random citizen). - $P(E|\bar{G}) = 10^{-6}$ (False positive rate). - $P(E|G) \approx 1$ (Sensitivity).

$$P(G | E) = \frac{P(E | G)P(G)}{P(E | G)P(G) + P(E | \bar{G})P(\bar{G})}$$

$$P(G | E) \approx \frac{1 \cdot 10^{-7}}{1 \cdot 10^{-7} + 10^{-6} \cdot 1} = \frac{10^{-7}}{1.1 \times 10^{-6}} \approx \frac{1}{11} \approx 0.09$$

Despite the “one in a million” match rarity, the probability of guilt is only about 9%! There are 10 million people, so we expect about 10 innocent matches ($10^7 \times 10^{-6}$) and 1 guilty match. Thus, out of 11 matches, only 1 is guilty. The prosecutor's argument ignores the base rate.

Example 2.9 (Island Problem). There are $N + 1$ people on the island and one is a criminal. We have probability of a trait of a criminal equal to p , which is $p = P(E | I)$, the probability of evidence, given innocence. Then we have a suspect who is matching the trait and we need to find probability of being guilty, given the evidence $P(G | E)$. It is easier to do the Bayes rule in the odds form. There are three components to the calculations: the prior odds of innocence,

$$O(I) = P(G)/P(I),$$

the Bayes factor,

$$\frac{P(E | G)}{P(E | I)}.$$

and the posterior odds of innocence.

$$O(I | E) = \frac{P(G | E)}{P(I | E)} = \frac{1}{Np}.$$

Cromwell's rule states that the use of prior probability of 1 or 0 should be avoided except when it is known for certain that the probability is 1 or 0. It is named after Oliver Cromwell who wrote to the General Assembly of the Church of Scotland in 1650 “*I beseech you, in the bowels of Christ, think it possible that you may be mistaken*”. In other words, using the Bayes rule

$$P(G | E) = \frac{P(E | G)}{P(E)} P(G),$$

if $P(G)$ is zero, it does not matter what the evidence is. Symmetrically, probability of innocence is zero if the evidence is certain. In other words, if $P(E | I) = 0$, then $P(I | E) = 0$. This is a very strong statement. It is not always true, but it is a good rule of thumb, it is a good way to avoid the prosecutor's fallacy.

Example 2.10 (Nakamura's Alleged Cheating). In our paper Maharaj, Polson, and Sokolov (2023), we provide a statistical analysis of the recent controversy between Vladimir Kramnik (ex-world champion) and Hikaru Nakamura. Kramnik called into question Nakamura's 45.5 out of 46 win streak in a 3+0 online blitz contest at chess.com. In this example we reproduce this paper and assess the weight of evidence using an a priori probabilistic assessment of Viswanathan Anand and the streak evidence of Kramnik. Our analysis shows that Nakamura has a 99.6 percent chance of not cheating given Anand's prior assumptions.

We start by addressing the argument of Kramnik which is based on the fact that the probability of such a streak is very small. This falls into precisely the **Prosecutor's Fallacy**, as discussed in Example 2.8. We denote by G the event of being guilty and I the event of innocence. We use E to denote evidence (streak of wins). Kramnik's argument is that because the probability of observing the streak is very low ($P(E|I)$ is small), it implies a high probability of cheating ($P(G|E)$ is high). As we have seen, this reasoning is flawed because it neglects the prior probability of cheating. Kramnik's calculations neglect other relevant factors, such as the prior probability of cheating. The prosecutor's fallacy can lead to an overestimation of the strength of the evidence and may result in an unjust conviction. In the cheating problem, at the top level of chess the prior probability of $P(G)$ is small! According to a recent statement by Viswanathan Anand, the probability of cheating is 1/10000.

News / Sports / Others / 'Online, it must be 1 in 10,000. But there are millions of g...
'Online, it must be 1 in 10,000. But there are millions of games': Viswanathan Anand addresses 'cheating' in chess

Figure 2.1: Anand's Prior

Given the prior ratio of cheaters to not cheaters is $1/N$, meaning out of $N + 1$ players, there is one cheater, the Bayes calculations require two main terms. The first one is the prior odds of guilt:

$$O(G) = P(I)/P(G).$$

Here $P(I)$ and $P(G)$ are the prior probabilities of innocence and guilt respectively.

The second term is the Bayes factor, which is the ratio of the probability of the evidence under the guilt hypothesis to the probability of the evidence under the innocence hypothesis. The Bayes factor is given by

$$L(E | G) = \frac{P(E | I)}{P(E | G)}.$$

The product of the Bayes factor and the prior odds is the posterior odds of guilt, given the evidence. The posterior odds of guilt is given by

$$O(G | E) = O(G) \times L(E | G).$$

The odds of guilt is

$$O(G) = \frac{N/(N+1)}{1/(N+1)} = N.$$

The Bayes factor is given by

$$\frac{P(E | I)}{P(E | G)} = \frac{p}{1} = p.$$

Thus, the posterior odds of guilt are

$$O(G | E) = Np.$$

There are two numbers we need to estimate to calculate the odds of cheating given the evidence, namely the prior probability of cheating given via N and the probability of a streak $p = P(E | I)$.

There are multiple ways to calculate the probability of a streak. We can use the binomial distribution, the negative binomial distribution, or the Poisson distribution. The binomial distribution is the most natural choice. The probability of a streak of k wins in a row is given by

$$P(E | I) = \binom{N}{k} q^k (1-q)^{N-k}.$$

Here q is the probability of winning a single game. Thus, for a streak of 45 wins in a row, we have $k = 45$ and $N = 46$. We encode the outcome of a game as 1 for a win and 0 for a loss or a draw. The probability of a win is $q = 0.8916$ (Nakamura's Estimate, he reported on his YouTube channel). The probability of a streak is then 0.029. The individual game win probability is calculated from the ELO rating difference between the players.

The ELO rating of Hikaru is 3300 and the average ELO rating of his opponents is 2950, according to Kramnik. The difference of 350 corresponds to the odds of winning of $wo = 10^{350/400} = 10^{0.875} = 7.2$. The probability of winning a single game is $q = wo/(1+wo) = 0.8916$.

Then we use Anand's prior of $N = 10000$ to get the posterior odds of cheating given the evidence of a streak of 45 wins in a row. The posterior odds of being innocent are 285. The probability of cheating is then

$$P(G | E) = 1/(1 + O(G | E)) = 0.003491.$$

Therefore the probability of innocence

$$P(I | E) = \frac{Np}{Np + 1} = 0.9965.$$

For completeness, we perform sensitivity analysis and also get the odds of not cheating for $N = 500$, which should be a high prior probability given the status of the player and the importance of the event. We get

$$P(I | E) = \frac{Np}{Np + 1} = 0.9445.$$

There are several assumptions we made in this analysis.

- Instead of calculating game-by-game probability of winning, we used the average probability of winning of 0.8916, provided by Nakamura himself. This is a reasonable assumption given the fact that Nakamura is a much stronger player than his opponents. This assumption slightly shifts posterior odds in favor of not cheating. Due to Jensen's inequality, we have $E(q^{50}) > E(q)^{50}$. Expected value of the probability of winning a single game is $E(q) = 0.8916$ and the expected value of the probability of a streak of 50 wins is $E(q^{50})$. We consider the difference between the two to be small. Further, there is some correlation between the games, which also shifts the posterior odds in favor of not cheating. For example, some players are on tilt. Given they lost the first game, they are more likely to lose the second game.
- There are many ways to win 3+0 unlike in classical chess. For example, one can win on time. We argue that the probability of winning calculated from the ELO rating difference is underestimated.

Next, we can use the Bayes analysis to solve an inverse problem and to find what prior you need to assume and how long of a sequence you need to observe to get 0.99 posterior? Small sample size, we have p close to 1. Figure 2.2 shows the combination of prior (N) and the probability of a streak (p) that gives posterior odds of 0.99.

Indeed, the results of the Bayesian analysis contradict the results of a traditional p-value based approach. A p-value is a measure used in frequentist statistical hypothesis testing. It represents the probability of obtaining the observed results, or results more extreme, assuming that the null hypothesis is true. The null hypothesis is a default position that Nakamura is not cheating

and we compare the ELO-based expected win probability of $q = 0.8916$ to the observed one of $s = 45/46 = 0.978$. Under the null hypothesis, Nakamura should perform at the level predicted by q .

```
q <- 0.8916
p <- dbinom(45, 46, q)
N <- 10000
odds <- p * N
print(1 - 1 / (1 + odds))
## 1
print(1 / (1 + odds))
## 0.0035
print(N * p / (N * p + 1))
## 1
```

```
p <- seq(from = 0.006, to = 0.07, length.out = 500)
N <- seq(500, 10000, by = 250)
plot(99 / N, N, xlab = "p", ylab = "N", type = "l", lwd = 3, col
+ = "blue")
```

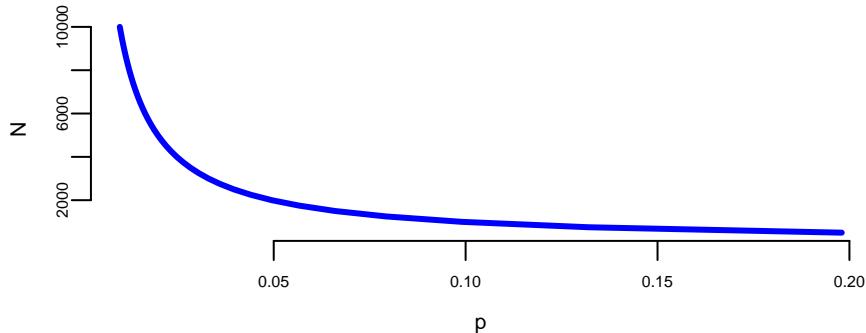


Figure 2.2: The combination of prior (N) and the probability of a streak (p) that gives posterior odds of 0.99.

David Hume discussed the problem similar to the Island problem in his “On Miracles” essay. Hume is making the following argument on miracles:

“...no testimony is sufficient to establish a miracle, unless the testimony be of such a kind, that its falsehood would be more miraculous, than the fact, which it endeavors to establish; and even in that case there is a mutual destruction of arguments, and the superior only gives us an assurance suitable to that degree of force, which remains, after deducting the inferior.”

One can view this as an application of the Island problem. Assuming the probability of a miracle A is $P(A) = p$ and $P(\text{not } A) = 1 - p$. Then Bayes rule gives

$$P(A|a) = \frac{P(a|A)p}{P(a|A)p + P(a|\text{not } A)(1-p)}$$

Prosecutor's fallacy, $P(a|\text{not } A) \neq 1 - P(a|A)$, in general.

In Hume's assessment of miracles (has to be something not in the laws of nature) we have $P(A) = 10^{-6}$. This assessment takes into account background information, I . Rare to have a contradiction to the laws of nature. More informative to write $P(A|I)$. Furthermore, we take $P(a|A) = 0.99$. The hardest bit is to assess $P(a|\text{not } A)$. The "frequency" of faked miracles and mankind's propensity to be marvelous. We assess $P(a|\text{not } A) = 10^{-3}$. This yields the chance of a miracle to be unlikely as

$$P(A|a) = \frac{0.99 \times 10^{-6}}{0.99 \times 10^{-6} + 10^{-3}(1 - 10^{-6})} \approx 10^{-3}.$$

Feynman considers the inverse problem: can we learn the laws of nature purely from empirical observation? Uses chess as an example. Is it a miracle that we have two bishops of the same color? No! according to Hume. We just didn't know the laws of nature (a.k.a. model).

Example 2.11 (Sally Clark Case: Independence or Bayes Rule?). To show that independence can lead to dramatically different results from Bayes conditional probabilities, consider the Sally Clark case. Sally Clark was accused and convicted of killing her two children who could have both died of SIDS. One explanation is that this was a random occurrence, the other one is that they both died of sudden infant death syndrome (SIDS). How can we use conditional probability to figure out a reasonable assessment of the probability that she murdered her children. First, some known probability assessments

1. The chance of a family of non-smokers having a SIDS death is 1 in 8,500.
2. The chance of a second SIDS death is 1 in 100.
3. The chance of a mother killing her two children is around 1 in 1,000,000.

Under Bayes

$$\begin{aligned} P(\text{both SIDS}) &= P(\text{first SIDS})P(\text{Second SIDS} | \text{first SIDS}) \\ &= \frac{1}{8500} \cdot \frac{1}{100} = \frac{1}{850,000}. \end{aligned}$$

The 1/100 comes from taking into account the genetic properties of SIDS. Independence, as implemented by the court, gets you to a probabilistic assessment

of

$$P(\text{both SIDS}) = (1/8500)(1/8500) = (1/73,000,000).$$

This is a low probability. It is still not the answer to our question of context. We need a conditional probability, this will come to the Bayes rule.

First, some general comment on the likelihood ratio calculation used to assess the weight of evidence in favor of guilty v.s. innocent evidence. Under Bayes we'll find that there's reasonable evidence that she'd be acquitted. We need the relative odds ratio. Let I denote the event that Sally Clark is innocent and G denotes guilty. Let E denote the evidence. In most cases, E contains a sequence E_1, E_2, \dots of 'facts' and we have to use the likelihood ratios in turn. Bayes rule then tells you to combine via multiplicative fashion. If likelihood ratio > 1 , odds of guilty. If likelihood ratio < 1 , more likelihood to be I . By Bayes rule

$$\frac{P(I | E)}{P(G | E)} = \frac{P(E \text{ and } I)}{P(E \text{ and } G)}.$$

If we further decompose $P(E \text{ and } I) = P(E | I)P(I)$ then we have to discuss the prior probability of innocence, namely $P(I)$. Hence this is one subtle advantage of the above decomposition.

The underlying intuition that Bayes gives us in this example, is that of the two possible explanations of the data, both of which are unlikely, it is the relative likelihood comparison that should matter. Here is a case where the p -value would be non-sensible ($P(E | I) \neq P(I | E)$). Effectively comparing two rare event probabilities from the two possible models or explanations.

Hence putting these two together gives the odds of guilt as

$$\frac{P(I | E)}{P(G | E)} = \frac{1/850,000}{1/1,000,000} = 1.15.$$

Solving for the posterior probability yields 46.5% for probability of guilty given evidence.

$$P(G | E) = \frac{1}{1 + O(G | E)} = 0.465.$$

Basically a 50/50 bet. Not enough to definitively convict! But remember that our initial prior probability on guilt $P(G)$ was 10^{-6} . So now there has been a dramatic increase to a posterior probability of 0.465. So it's not as if Bayes rule thinks this is evidence in the suspect's favor – but the magnitude is still not in the 0.999 range though, where most jurors would have to be to feel comfortable with a guilt verdict.

If you use the "wrong" model of independence (as the court did) you get

$$P(\text{both SIDS}) = \frac{1}{8500} \cdot \frac{1}{8500} = \frac{1}{73,000,000}.$$

With the independence assumption, you make the assessment

$$\frac{P(I | E)}{P(G | E)} = \frac{1}{73} \text{ and } P(G | E) \approx 0.99.$$

Given these probability assumptions, the suspect looks guilty with probability 99%.

Experts also mis-interpret the evidence by saying: *1 in 73 million chance that it is someone else*. This is clearly false and misleading to the jury and has led to appeals.

Example 2.12 (O. J. Simpson Case: Dershowitz Fallacy).

This example is based on I. J. Good's, "When batterer turns murderer." *Nature*, 15 June 1995, p. 541. Alan Dershowitz, on the O. J. Simpson defense team, stated on T.V. and in newspapers that only 1 in 2,500 of men who abuse their wives go on to murder them. He clearly wanted his audience to interpret this to mean that the evidence of abuse by Simpson would only suggest a 0.04% probability of his being guilty of murdering her. He used probability to argue that because so few husbands who batter their wives actually go on to murder their wives. Thus, O.J. is highly likely to be not guilty. This leaves out the most relevant conditioning information that we also know that Nicole Brown Simpson was actually murdered. Both authors believe the jury would be more interested in the probability that the husband is guilty of the murder of his wife given that he abused his wife and his wife was murdered. They both solve this problem by using Bayes' theorem.

In this example, the notation B represents "woman battered by her husband, boyfriend, or lover", M represents the event "woman murdered", and G denotes "woman murdered by her batterer". Our goal is to show that

$$P(G | M, B) \neq P(G | B).$$

It is not hard to come to a wrong conclusion if you don't take into account all the relevant conditional information. He intended this information to exonerate O.J. In 1992 the women population of the US was 125 million and 4936 women were murdered, thus

$$P(M) = 4936/125,000,000 = 0.00004 = 1/25,000.$$

At the same year about 3.5 million women were battered

$$P(B) = 3.5/125 = 0.028.$$

That same year 1432 women were murdered by their previous batterers, so the marginal probability of that event is $P(G) = 1432/125,000,000 = 0.00001 = 1/87,290$, and the conditional probability, $P(G|B)$ is 1432 divided by 3.5 million, or $1/2444$. These are the numbers Dershowitz used to obtain his estimate that about 1 in 2500 battered women go on to be murdered by their batterers.

We need to calculate

$$P(G | M, B) = P(M|G, B)P(G)/P(M).$$

We know $P(M|G, B) = 1$ and $P(G)/P(M) = 0.00001/0.00004 = 0.29$, or about 1 in 3.5.

Alan Dershowitz provided the jury with an accurate but irrelevant probability. The fact the woman was murdered increases the probability that she was murdered by her batterer by a factor of 709 ($0.29/(1/2444)$).

$$P(G | M, B) \approx 709 \times P(G | B).$$

The argument used by Dershowitz relating to the Simpson case has been discussed by John Paulos in an op-ed article in the Philadelphia Inquirer (15 Oct. 1995, C7) and his book “Once Upon a Number”, by I.J. Good in an article in Nature (June 15, 1995, p 541) and by Jon Merz and Jonathan Caulkins in an article in Chance Magazine, (Spring 1995, p 14).

2.4 First Application: Naive Bayes

Use of the Bayes rule allows us to build our first predictive model, called Naive Bayes classifier. Naive Bayes is a collection of classification algorithms based on Bayes Theorem. It is not a single algorithm but a family of algorithms that all share a common principle, that every feature being classified is independent of the value of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3” in diameter. A Naive Bayes classifier considers each of these “features” (red, round, 3” in diameter) to contribute independently to the probability that the fruit is an apple, regardless of any correlations between features. Features, however, aren’t always independent which is often seen as a shortcoming of the Naive Bayes algorithm and this is why it’s labeled “naive”.

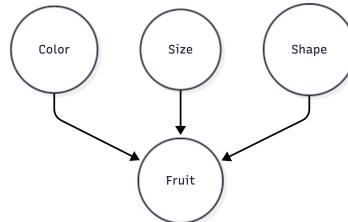


Figure 2.3: Color, Shape and Size of Fruits

Although it's a relatively simple idea, Naive Bayes can often outperform other more sophisticated algorithms and is extremely useful in common applications like spam detection and document classification. In a nutshell, the algorithm allows us to predict a class, given a set of features using probability. So in another fruit example, we could predict whether a fruit is an apple, orange or banana (class) based on its colour, shape etc (features). In summary, the advantages are:

- It's relatively simple to understand and build
- It's easily trained, even with a small dataset
- It's fast!
- It's not sensitive to irrelevant features

The main disadvantage is that it assumes every feature is independent, which isn't always the case.

Let's say we have data on 1000 pieces of fruit. The fruit being a Banana, Orange or some Other fruit and imagine we know 3 features of each fruit, whether it's long or not, sweet or not and yellow or not, as displayed in the table below:

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	500	650	800	1000

From this data we can calculate marginal probabilities

- 50% of the fruits are bananas
- 30% are oranges
- 20% are other fruits

Based on our training set we can also say the following:

- From 500 bananas 400 (0.8) are Long, 350 (0.7) are Sweet and 450 (0.9) are Yellow
- Out of 300 oranges 0 are Long, 150 (0.5) are Sweet and 300 (1) are Yellow
- From the remaining 200 fruits, 100 (0.5) are Long, 150 (0.75) are Sweet and 50 (0.25) are Yellow So let's say we're given the features of a piece of fruit and we need to predict the class. If we're told that the additional fruit is Long, Sweet and Yellow, we can classify it using the following formula and subbing in the values for each outcome, whether it's a Banana, an Orange or Other Fruit. The one with the highest probability (score) being the winner.

Given the evidence E ($L = \text{Long}$, $S = \text{Sweet}$ and $Y = \text{Yellow}$) we can calculate the probability of each class C ($B = \text{Banana}$, $O = \text{Orange}$ or $F = \text{Other Fruit}$) using Bayes' Theorem:

$$\begin{aligned} P(B | E) &= \frac{P(L | B)P(S | B)P(Y | B)P(B)}{P(L)P(S)P(Y)} \\ &= \frac{0.8 \times 0.7 \times 0.9 \times 0.5}{P(E)} = \frac{0.252}{P(E)} \end{aligned}$$

Orange:

$$P(O | E) = 0.$$

Other Fruit:

$$\begin{aligned} P(F | E) &= \frac{P(L | F)P(S | F)P(Y | F)P(F)}{P(L)P(S)P(Y)} \\ &= \frac{0.5 \times 0.75 \times 0.25 \times 0.2}{P(E)} = \frac{0.01875}{P(E)} \end{aligned}$$

In this case, based on the higher score, we can assume this Long, Sweet and Yellow fruit is, in fact, a Banana.

Notice, we did not have to calculate $P(E)$ because it is a normalizing constant and it cancels out when we calculate the ratio

$$\frac{P(B | E)}{P(F | E)} = 0.252 / 0.01875 = 13.44 > 1.$$

Now that we've seen a basic example of Naive Bayes in action, you can easily see how it can be applied to Text Classification problems such as spam detection, sentiment analysis and categorization. By looking at documents as a set of words, which would represent features, and labels (e.g. "spam" and "ham" in case of spam detection) as classes we can start to classify documents and text automatically.

Example 2.13 (Spam Filtering). The original spam filtering algorithm was based on Naive Bayes. The "naive" aspect of Naive Bayes comes from the assumption that inputs (words in the case of text classification) are conditionally independent, given the class label. Naive Bayes treats each word independently, and the model doesn't capture the sequential or structural information inherent in the language. It does not consider grammatical relationships or syntactic structures. The algorithm doesn't understand the grammatical rules that dictate how words should be combined to form meaningful sentences. Further, it doesn't understand the context in which words appear. For example, it may treat the word "bank" the same whether it refers to a financial institution or the side of a river bank. Despite its simplicity and the naive

assumption, Naive Bayes often performs well in practice, especially in text classification tasks.

We start by collecting a dataset of emails labeled as “spam” or “not spam” (ham) and calculate the prior probabilities of spam ($P(\text{spam})$) and not spam ($P(\text{ham})$) based on the training dataset, by simply counting the proportions of each in the data.

Then each email gets converted into a bag-of-words representation (ignoring word order and considering only word frequencies). Then, we create a vocabulary of unique words from the entire dataset w_1, w_2, \dots, w_N and calculate conditional probabilities

$$P(\text{word}_i | \text{spam}) = \frac{\text{Number of spam emails containing word}_i}{\text{Total number of spam emails}}, \quad i = 1, \dots, n$$

$$P(\text{word}_i | \text{ham}) = \frac{\text{Number of ham emails containing word}_i}{\text{Total number of ham emails}}, \quad i = 1, \dots, n$$

Now, we are ready to use our model to classify new emails. We do it by calculating the posterior probability using Bayes’ theorem. Say an email has a set of k words email = $\{w_{e1}, w_{e2}, \dots, w_{ek}\}$, then

$$P(\text{spam} | \text{email}) = \frac{P(\text{email} | \text{spam}) \times P(\text{spam})}{P(\text{email})}$$

Here

$$P(\text{email} | \text{spam}) = P(w_{e1} | \text{spam})P(w_{e2} | \text{spam}) \dots P(w_{ek} | \text{spam})$$

We calculate $P(\text{ham} | \text{email})$ in a similar way.

Finally, we classify the email as spam or ham based on the class with the highest posterior probability.

Suppose you have a spam email with the word “discount” appearing. Using Naive Bayes, you’d calculate the probability that an email containing “discount” is spam $P(\text{spam} | \text{discount})$ and ham $P(\text{ham} | \text{discount})$, and then compare these probabilities to make a classification decision.

While the naive assumption simplifies the model and makes it computationally efficient, it comes at the cost of a more nuanced understanding of language. More sophisticated models, such as transformers, have been developed to address these limitations by considering the sequential nature of language and capturing contextual relationships between words.

In summary, naive Bayes, due to its simplicity and the naive assumption of independence, is not capable of understanding the rules of grammar, the order of words, or the intricate context in which words are used. It is a basic algorithm suitable for certain tasks but may lack the complexity needed for tasks that require a deeper understanding of language structure and semantics.

2.5 Sensitivity and Specificity

Conditional probabilities are used to define two fundamental metrics used for many probabilistic and statistical learning models, namely *sensitivity* and *specificity*.

Sensitivity and specificity are two key metrics used to evaluate the performance of diagnostic tests, classification models, or screening tools. These metrics help assess how well a test can correctly identify individuals with a condition (true positives) and those without the condition (true negatives). Let's break down each term:

1. *Sensitivity (true-positive rate or recall)* is the ability of a test T to correctly identify individuals who have a particular condition or disease (D), $P(T = 1 | D = 1)$, the probability of a positive test given that the individual has the disease. It is calculated as the ratio of true positives to the sum of true positives and false negatives.

$$P(T = 1 | D = 1) = \frac{P(T = 1, D = 1)}{P(D = 1)}.$$

A high sensitivity indicates that the test is good at identifying individuals with the condition, minimizing false negatives.

2. *Specificity (true-negative rate)* is the ability of a test to correctly identify individuals who do not have a particular condition or disease, $P(T = 0 | D = 0)$. It is calculated as the ratio of true negatives to the sum of true negatives and false positives.

$$P(T = 0 | D = 0) = \frac{P(T = 0, D = 0)}{P(D = 0)}$$

A high specificity indicates that the test is good at correctly excluding individuals without the condition, minimizing false positives.

Sensitivity and specificity are often trade-offs. Increasing sensitivity might decrease specificity, and vice versa. Thus, depending on the application, you might prefer sensitivity over specificity or vice versa, depending on the consequences of false positives and false negatives in a particular application.

Consider a medical test designed to detect a certain disease. If the test has high sensitivity, it means that it is good at correctly identifying individuals with the disease. On the other hand, if the test has high specificity, it is good at correctly identifying individuals without the disease. The goal is often to strike a balance between sensitivity and specificity based on the specific needs and implications of the test results.

Sensitivity is often called the power of a procedure (a.k.a. test). Type I and Type II errors are fundamental concepts in hypothesis testing, serving as the duals to specificity and sensitivity.

Type I error (false positive rate)

is the percentage of healthy people who tested positive, $P(T = 1 | D = 0)$, it is the mistake of thinking something is true when it is not.

Type II error (or false negative rate)

is the percentage of sick people who are tested negative, $P(T = 0 | D = 1)$, it is the mistake of thinking something is not true when in fact it is true.

We would like to control both conditional probabilities with our test. Also if someone tests positive, how likely is it that they actually have the disease. There are two ‘errors’ one can make. Falsely diagnosing someone, or not correctly finding the disease.

In the stock market, one can think of type I error as not selling a losing stock quickly enough, and a type II error as failing to buy a growing stock, e.g. Amazon or Google.

Table 2.8: Sensitivity and Specificity

$P(T = 1 D = 1)$	Sensitivity	True Positive Rate	$1 - \beta$
$P(T = 0 D = 0)$	Specificity	True Negative Rate	$1 - \alpha$
$P(T = 1 D = 0)$	1-Specificity	False Positive Rate	α (type I error)
$P(T = 0 D = 1)$	1-Sensitivity	False Negative Rate	β (type II error)

Often it is convenient to write those four values in the form of a two-by-two matrix, called the confusion matrix:

Actual/Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

where: TP: True Positive. FN: False Negative, FP: False Positive, TN: True Negative

We will extensively use the concepts of errors, specificity and sensitivity later in the book, when describing AB testing and predictive models. These examples illustrate why people can commonly miscalculate and mis-interpret probabilities. Those quantities can be calculated using the Bayes rule.

2.5.1 Medical Diagnostics

Example 2.14 (Alice Mammogram). Alice is a 40-year-old woman, what is the chance that she really has breast cancer when she gets positive mammogram result, given the conditions:

1. The prevalence of breast cancer among people like Alice is 1%.
2. The test has an 80% detection rate.
3. The test has a 10% false-positive rate.

We want to calculate the posterior probability $P(\text{cancer} \mid \text{positive mammogram})$.

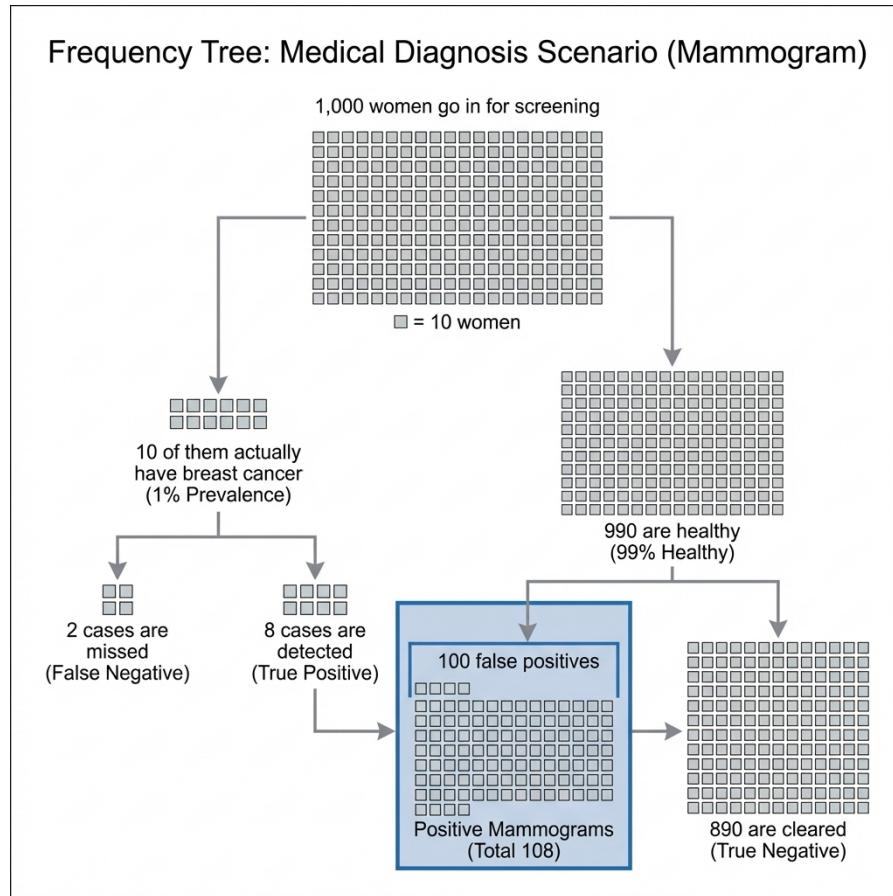


Figure 2.4: Frequency Tree: Medical Diagnosis Scenario (Mammogram)

Using the frequency tree in Figure 2.4, we can see that out of 1000 cases:

- Number of actual cancer cases = 10.
- Number of healthy cases = 990.

The test detects 8 out of the 10 cancer cases (True Positives). The test falsely flags 100 out of the 990 healthy cases (False Positives). The total number of positive mammograms is thus $8 + 100 = 108$. The number of these that are actually cancer is 8. Therefore, the posterior probability is:

$$P(\text{cancer} | \text{positive}) = \frac{8}{108} \approx 0.074.$$

There is only about a 7.4% chance Alice has cancer, despite the positive test result.

Example 2.15 (Apple Watch Series 4 ECG and Bayes' Theorem). The Apple Watch Series 4 can perform a single-lead ECG and detect atrial fibrillation. The software can correctly identify 98% of cases of atrial fibrillation (true positives) and 99% of cases of non-atrial fibrillation (true negatives) (Kim et al. 2024; Bumgarner et al. 2018).

Predicted	atrial fibrillation	no atrial fibrillation	Total
atrial fibrillation	1960	980	2940
no atrial fibrillation	40	97020	97060
Total	2000	98000	100000

However, what is the probability of a person having atrial fibrillation when atrial fibrillation is identified by the Apple Watch Series 4? We use Bayes theorem to answer this question.

$$P(\text{atrial fibrillation} \mid \text{atrial fibrillation is identified}) = \frac{0.01960}{0.02940} = 0.6667$$

The conditional probability of having atrial fibrillation when the Apple Watch Series 4 detects atrial fibrillation is about 67%.

Apple Watch's positive predictive value is just 19.6 percent. That means in this group – which constitutes more than 90 percent of users of wearable devices like the Apple Watch – the app incorrectly diagnoses atrial fibrillation 79.4 percent of the time. (You can try the calculation yourself using this Bayesian calculator: enter 0.02 for prevalence, 0.98 for sensitivity, and 0.99 for specificity).

The electrocardiogram app becomes more reliable in older individuals: The positive predictive value is 76 percent among users between the ages of 60 and 64, 91 percent among those aged 70 to 74, and 96 percent for those older than 85.

In the case of medical diagnostics, the sensitivity is the ratio of people who have disease and tested positive to the total number of positive cases in the population

$$P(T = 1 \mid D = 1) = \frac{P(T = 1, D = 1)}{P(D = 1)} = 0.0196/0.02 = 0.98$$

The specificity is given by

$$P(T = 0 \mid D = 0) = \frac{P(T = 0, D = 0)}{P(D = 0)} = 0.9702/0.98 = 0.99.$$

As we see the test is highly sensitive and specific. However, only 66% of those who are tested positive will have a disease. This is due to the fact that the number of sick people is much less than the number of healthy and presence of type I error.

2.6 Advanced Applications

Example 2.16 (Obama Elections). This example demonstrates a Bayesian approach to election forecasting using polling data from the 2012 US presidential election. The goal is to predict the probability of Barack Obama winning the election by combining polling data across different states.

The data used includes polling data from various pollsters across all 50 states plus DC. Each state has polling percentages for Republican (GOP) and Democratic (Dem) candidates along with their electoral vote counts. The data is aggregated by state, taking the most recent polls available.

The techniques applied involve Bayesian simulation using a Dirichlet distribution to model uncertainty in polling percentages.

Dirichlet Distribution

The Dirichlet distribution is a distribution *over probabilities*. For a K -dimensional vector $\mathbf{x} = (x_1, \dots, x_K)$ such that $\sum_{i=1}^K x_i = 1$ and $x_i \geq 0$, the probability density function is:

$$f(\mathbf{x}; \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^K x_i^{\alpha_i - 1}$$

where $\alpha = (\alpha_1, \dots, \alpha_K)$ are positive parameters and $B(\alpha) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}$ is the multivariate Beta function. It is used here to sample possible election results (Obama, Romney, Others) that always sum to 100%.

Monte Carlo simulation runs 10,000 simulations of the election to estimate win probabilities. The analysis is conducted state-by-state, calculating Obama's probability of winning each individual state. Electoral college modeling combines state probabilities with electoral vote counts to determine the overall election outcome. The simulation runs the entire election multiple times to account for uncertainty and determines the likelihood of Obama reaching the required 270 electoral votes to win. This approach demonstrates how pattern matching through statistical modeling can be used for prediction, showing how polling data can be transformed into probabilistic forecasts of election outcomes.

We start by loading the data and aggregating it by state. We then run the simulation and plot probabilities by state.

We then plot the probabilities of Obama winning by state.

state	R.pct	O.pct	EV	state	R.pct	O.pct	EV
Alabama	61	38	9	47	Virginia	48	51 13
Alaska	55	42	3	48	Washington	42	56 12
Arizona	54	44	11	49	West Virginia	62	36 5
Arkansas	61	37	6	50	Wisconsin	46	53 10
California	38	59	55	51	Wyoming	69	28 3

Election 2012 Data (first 5 states and last 5 states)

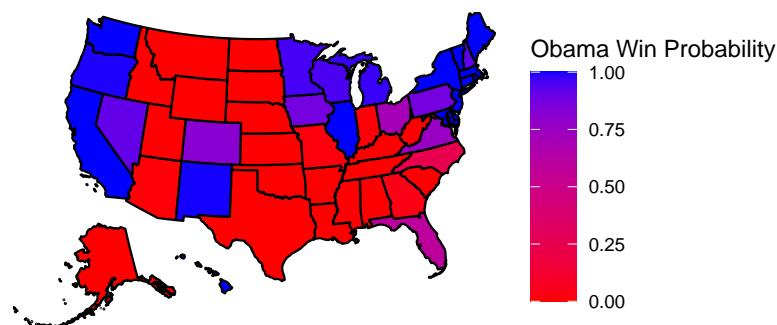


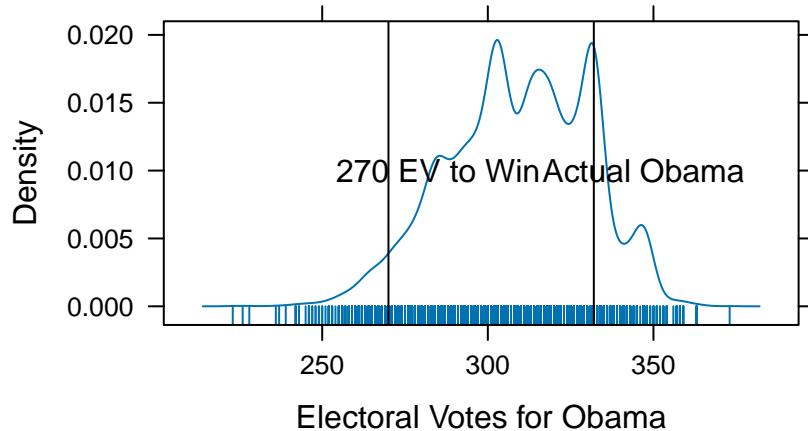
Figure 2.5: Probabilities of Obama winning by state

We use those probabilities to simulate the probability of Obama winning the election. First, we calculate the probability of Obama having 270 EV or more

```
sim.election <- function(win.probs) {
  winner <- rbinom(51, 1, win.probs$Obama)
  sum(win.probs$EV * winner)
}

sim.EV <- replicate(10000, sim.election(win.probs))
oprob <- sum(sim.EV >= 270) / length(sim.EV)
oprob
## 0.97
```

Electoral College Results Probability



Results of recent state polls in the 2008 United States Presidential Election between Barack Obama and John McCain.

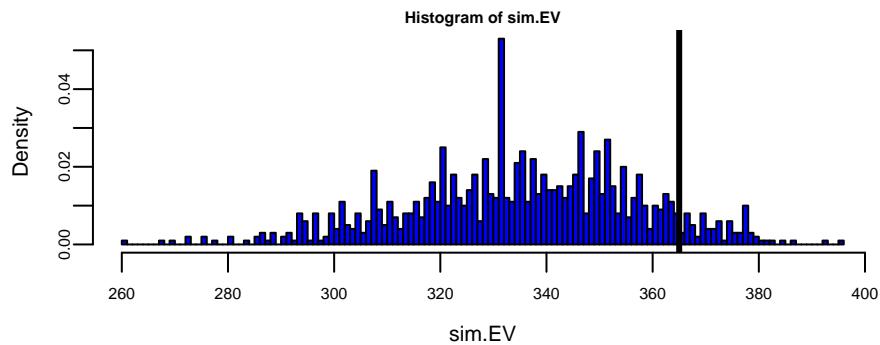


Figure 2.6: Histogram of simulated election

The analysis of the 2008 U.S. Presidential Election data reveals several key insights about the predictive power of state-level polling and the uncertainty inherent in electoral forecasting. The actual result of 365 electoral votes falls within the simulated range, demonstrating the model's validity. The 270-vote threshold needed to win the presidency is clearly marked and serves as a critical reference point.

We used a relatively simple model to simulate the election outcome. The model uses Dirichlet distributions to capture uncertainty in state-level polling percentages. Obama's win probabilities vary significantly across states, reflecting

the competitive nature of the election. The simulation approach accounts for both sampling uncertainty and the discrete nature of electoral vote allocation. The histogram of simulated results shows the distribution of possible outcomes. The actual Obama total of 365 electoral votes is marked and falls within the reasonable range of simulated outcomes. This validates the probabilistic approach to election forecasting.

This analysis demonstrates how Bayesian methods can be effectively applied to complex prediction problems with multiple sources of uncertainty, providing both point estimates and uncertainty around those estimates.

2.7 Graphical Representation of Conditional Independence.

We can use the telescoping property of conditional probabilities to write the joint probability distribution as a product of conditional probabilities. This is the essence of the chain rule of probability. It is given by

$$P(x_1, x_2, \dots, x_n) = P(x_1)P(x_2 | x_1)P(x_3 | x_1, x_2) \dots P(x_n | x_1, x_2, \dots, x_{n-1}).$$

The expression on the right hand side can be simplified if some of the variables are conditionally independent. For example, if x_3 is conditionally independent of x_2 , given x_1 , then we can write

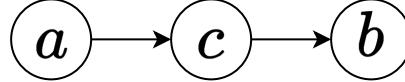
$$P(x_3 | x_1, x_2) = P(x_3 | x_1).$$

In a high-dimensional case, when we have a joint distribution over a large number of random variables, we can often simplify the expression by using independence or conditional independence assumptions. Sometimes it is convenient to represent these assumptions in a graphical form. This is the idea behind the concept of a Bayesian network. Essentially, the graph is a compact representation of a set of independencies that hold in the distribution.

Let's consider an example of joint distribution with three random variables, we have the following joint distribution:

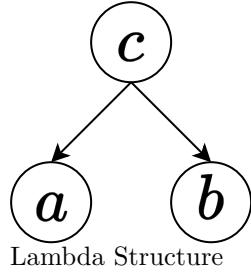
$$P(a, b, c) = P(a | b, c)P(b | c)P(c)$$

Graphically, we can represent the relations between the variables known as a Directed Acyclic Graph (DAG), which is known as a Bayesian network. Each node represents a random variable and the arrows represent the conditional dependencies between the variables. When two nodes are connected they are not independent. Consider the following three cases:



Line Structure

$$P(b | c, a) = P(b | c), P(a, b, c) = P(a)P(c | a)P(b | c)$$



Lambda Structure

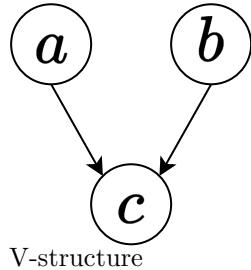
$$P(a | b, c) = P(a | c), P(a, b, c) = P(a | c)P(b | c)P(c)$$

Although the graph shows us the conditional independence assumptions, we can also derive other independencies from the graph. An interesting question is whether they are connected through a third node. In the first case (a), we have a and b connected through c . Thus, a can influence b . However, once c is known, a and b are independent. In case (b) the logic here is similar, a can influence b through c , but once c is known, a and b are independent. In the third case (c), a and b are independent, but once c is known, a and b are not independent. You can formally derive these independencies from the graph by comparing $P(a, b | c)$ and $P(a | c)P(b | c)$.

Example 2.17 (Bayes Home Diagnostics). Suppose that a house alarm system sends me a text notification when some motion inside my house is detected. It detects motion when I have a person inside (burglar) or during an earthquake. Say, from prior data we know that during an earthquake alarm is triggered in 10% of the cases. Once I receive a text message, I start driving back home. While driving I hear on the radio about a small earthquake in our area. Now we want to know $P(b | a)$ and $P(b | a, r)$. Here b = burglary, e = earthquake, a = alarm, and r = radio message about small earthquake.

The joint distribution is then given by

$$P(b, e, a, r) = P(r | a, b, e)P(a | b, e)P(b | e)P(e).$$



V-structure

$$P(a | b) = P(a), P(a, b, c) = P(c | a, b)P(a)P(b)$$

Since we know the causal relations, we can simplify this expression

$$P(b, e, a, r) = P(r | e)P(a | b, e)P(b)P(e).$$

The $P(a | b, e)$ distribution is defined by

Table 2.13: Conditional probability of alarm given burglary and earthquake

$P(a = 1 b, e)$	b	e
0	0	0
0.1	0	1
1	1	0
1	1	1

Graphically, we can represent the relations between the variables known as a Directed Acyclic Graph (DAG), which is known as a Bayesian network.

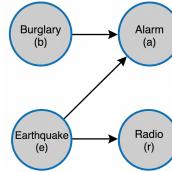


Figure 2.7: Bayesian network for alarm.

Now we can easily calculate $P(a = 0 | b, e)$, from the property of a probability distribution $P(a = 1 | b, e) + P(a = 0 | b, e) = 1$. In addition, we are given $P(r = 1 | e = 1) = 0.5$ and $P(r = 1 | e = 0) = 0$. Further, based on historic data we have $P(b) = 2 \cdot 10^{-4}$ and $P(e) = 10^{-2}$. Note that causal relations allowed us to have a more compact representation of the joint probability distribution. The original naive representation requires specifying 2^4 parameters.

To answer our original question, calculate

$$P(b | a) = \frac{P(a | b)P(b)}{P(a)}, \quad P(a) = P(a = 1 | b = 1)P(b = 1) + P(a = 1 | b = 0)P(b = 0).$$

We have everything but $P(a | b)$. This is obtained by marginalizing $P(a = 1 | b, e)$, to yield

$$P(a | b) = P(a | b, e = 1)P(e = 1) + P(a | b, e = 0)P(e = 0).$$

We can calculate

$$P(a = 1 | b = 1) = 1, \quad P(a = 1 | b = 0) = 0.1 * 10^{-2} + 0 = 10^{-3}.$$

This leads to $P(b | a) = 2 \cdot 10^{-4} / (2 \cdot 10^{-4} + 10^{-3}(1 - 2 \cdot 10^{-4})) = 1/6$.

This result is somewhat counterintuitive. We get such a low probability of burglary because its prior is very low compared to the prior probability of an earthquake. What will happen to the posterior if we live in an area with higher crime rates, say $P(b) = 10^{-3}$. Figure 2.8 shows the relationship between the prior and posterior.

$$P(b | a) = \frac{P(b)}{P(b) + 10^{-3}(1 - P(b))}$$

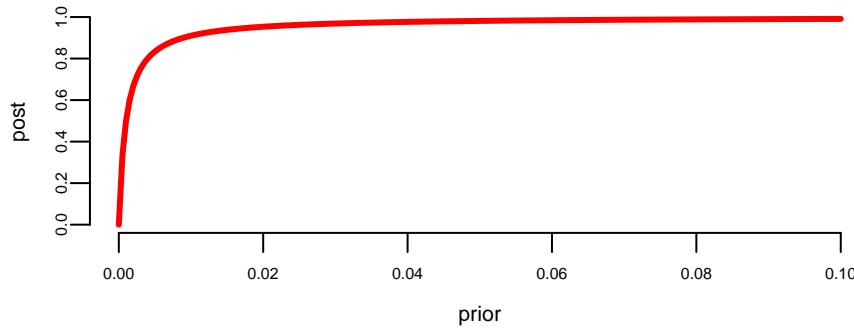


Figure 2.8: Relationship between the prior and posterior

Now, suppose that you hear on the radio about a small earthquake while driving. Then, using Bayesian conditioning,

$$P(b = 1 | a = 1, r = 1) = \frac{P(a, r | b)P(b)}{P(a, r)}$$

and

$$\begin{aligned} P(a, r | b)P(b) &= \frac{\sum_e P(b = 1, e, a = 1, r = 1)}{\sum_b \sum_e P(b, e, a = 1, r = 1)} \\ &= \frac{\sum_e P(r = 1 | e)P(a = 1 | b = 1, e)P(b = 1)P(e)}{\sum_b \sum_e P(r = 1 | e)P(a = 1 | b, e)P(b)P(e)} \end{aligned}$$

which is $\approx 2\%$ in our case. This effect is called *explaining away*, namely when new information explains some previously known fact.

3

Bayesian Learning

“A Wise man proportions his beliefs to the evidence.” – David Hume

Statistics makes use of parametric families of distributions and often assumes that observed samples $y = (y_1, \dots, y_n)$ are independent and identically distributed observations from a distribution with density function parametrized by θ , the notation is $y | \theta \sim p(y | \theta)$. The functional form of $p(y | \theta)$ is assumed to be known, but the value of θ is unknown. The goal of statistical inference is to estimate θ from the observed data

$$y = (y_1, \dots, y_n).$$

The main tasks in statistical inference include estimation, hypothesis testing, and prediction. **Estimation** involves using the observed sample to infer the value of the unknown parameter θ , either by providing a single best guess (denoted as $\hat{\theta}$) or by constructing an interval $[a, b]$ that is likely to contain the true value of θ with a specified probability. **Hypothesis testing** focuses on evaluating specific claims or hypotheses about the value of θ ; for instance, we might be interested in determining whether θ is equal to a particular value θ_0 . **Prediction**, on the other hand, is concerned with forecasting the value of a future observation y_{n+1} based on the data already observed, y_1, \dots, y_n via a model $p(y_{n+1} | y_1, \dots, y_n)$.

In this section we present a general framework for statistical inference, known as Bayesian inference, which is based on the use of probability distributions to represent uncertainty and make inferences about unknown parameters. We will use Bayes rule to update our beliefs about the parameters of a model based on new evidence or data. Bayesian inference provides a principled approach to statistical modeling and decision-making.

Bayesian analysis always starts with specifying the two main components of the model:

1. **Prior** $p(\theta)$ represents your beliefs or knowledge about the parameters before observing any data. This distribution encapsulates the uncertainty about the parameters.

2. **Likelihood Function (Data Likelihood):** $p(y | \theta)$ describes the likelihood of observing the given data given the current values of the parameters. When we have independent and identically distributed $y = (y_1, \dots, y_n)$, the likelihood function is given by

$$p(y | \theta) = \prod_{i=1}^n p(y_i | \theta).$$

The estimation problem is also called Bayesian parameter learning; the goal is then to update the probability distribution over the parameters of the model as new data becomes available. Suppose that you are interested in the values of k unknown quantities

$$\theta = (\theta_1, \dots, \theta_k).$$

Then learning is done by updating the prior distribution over the parameters of the model as new data becomes available and calculating the posterior distribution over the parameters. The posterior distribution represents the updated beliefs about the parameters after incorporating the observed data. It combines the prior distribution and the likelihood function using Bayes' theorem:

$$p(\theta | y) = \frac{p(y | \theta)p(\theta)}{p(y)} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Marginal}}.$$

The left hand side $p(\theta|y)$ is the posterior distribution, and $p(y)$ is the probability of the observed data (also known as the total probability) given by

$$p(y) = \int p(y | \theta)p(\theta)d\theta$$

Nice feature of the Bayes approach is that you can use the posterior distribution obtained from one round of observation as the prior distribution for the next round when more data becomes available. This process can be iteratively repeated as new evidence is acquired. It makes sequential learning possible.

The downstream tasks then use the posterior distribution. For example, if we want to predict the value of a future observation y_{n+1} based on the data already observed, y_1, \dots, y_n via a model $p(y_{n+1} | y_1, \dots, y_n, \theta)$, we can use the posterior distribution to calculate the expected value of y_{n+1} as

$$p(y_{n+1} | y_1, \dots, y_n) = \int p(y_{n+1} | y_1, \dots, y_n, \theta)p(\theta | y_1, \dots, y_n)d\theta.$$

You can think of prediction as a mixture of the conditional distributions $p(y_{n+1} | y_1, \dots, y_n, \theta)$ weighted by the posterior distribution $p(\theta | y_1, \dots, y_n)$.

A simple hypothesis test then involves calculating the probability of the observed data under the null hypothesis H_0 as

$$p(y | H_0) = \int p(y | \theta)p(\theta | H_0)d\theta.$$

The downside of the Bayes approach is the requirement to calculate the marginal likelihood $p(y)$, which often requires calculating high-dimensional integrals that are intractable (do not have closed-form solutions). However, often, Bayesian analysis can be performed without calculating the marginal likelihood, in this case we omit the total probability in the denominator on the right hand side and write Bayes rule as

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}.$$

The choice of prior distribution can significantly impact the ease of computation and the interpretation of the posterior distribution. Conjugate priors are a special type of prior distribution that, when combined with a specific likelihood function, result in a posterior distribution that belongs to the same family as the prior. This property simplifies the computation of the posterior distribution, and allows for analytical solutions.

Common examples of conjugate priors include:

- *Normal distribution with known variance:* If the likelihood is a normal distribution with known variance, then a normal distribution is a conjugate prior for the mean.
- *Binomial distribution:* If the likelihood is a binomial distribution, then a beta distribution is a conjugate prior for the probability of success.
- *Poisson distribution:* If the likelihood is a Poisson distribution, then a Gamma distribution is a conjugate prior for the rate parameter.

Using conjugate priors simplifies the Bayesian analysis, especially in cases where analytical solutions are desirable. However, the choice of a conjugate prior is often a modeling assumption, and in some cases, non-conjugate priors may be more appropriate for capturing the true underlying uncertainty in the problem. The blind use of conjugate priors can lead to misleading results. We should never ignore the absence of evidence for use of a specific model.

3.1 Exchangeability and the Bayesian view of probability models

At the basis of all statistical problems is a potential sample of data, $y = (y_1, \dots, y_T)$, and assumptions over the data generating process such as independence, a model or models, and parameters. How should one view the relationship between models, parameters, and samples of data? How should one

define a model and parameters? These questions have fundamental implications for statistical inference and can be answered from different perspectives. We will discuss de Finetti's representation theorem which provides a formal connection between data, models, and parameters.

To understand the issues, consider the simple example of an experiment consisting of tosses of a simple thumb tack in ideal “laboratory” conditions.¹ The outcome of the experiment can be defined as a random variable y_i , where $y_i = 1$ if the i^{th} toss was a heads (the tack lands on the spike portion) and $y_i = 0$ if the tack lands tails (on its flat portion). How do we model these random variables? The frequentist or objective approach assumes tosses are independent and identically distributed. In this setting, independence implies that

$$P(y_2 = 1, y_1 = 1) = P(y_2 = 1)P(y_1 = 1).$$

Given this, are thumbtack tosses independent? Surprisingly, the answer is no. Or at least absolutely not under the current assumptions. Independence implies that

$$P(y_2 = 1 \mid y_1 = 1) = P(y_2 = 1),$$

which means that observing $y_1 = 1$ does not affect the probability that $y_2 = 1$. To see the implications of this simple fact, suppose that the results of 500 tosses were available. If the tosses were independent, then

$$P(y_{501} = 1) = P\left(y_{501} = 1 \mid \sum_{t=1}^{500} y_t = 1\right) = P\left(y_{501} = 1 \mid \sum_{t=1}^{500} y_t = 499\right).$$

It is hard to imagine that anyone would believe this—nearly every observer would state that the second probability is near zero and the third probability is near 1 as the first 500 tosses contain a lot of information. Thus, the tosses are not independent.

To see the resolution of this apparent paradox, introduce a parameter, θ , which is the probability that a thumb tack toss is heads. If θ were known, then it is true that, conditional on the value of this parameter, the tosses are independent and

$$P(y_2 = 1 \mid y_1 = 1, \theta) = P(y_2 = 1 \mid \theta) = \theta.$$

Thus, the traditional usage of independence, and independent sampling, requires that “true” parameter values are known. With unknown probabilities, statements about future tosses are heavily influenced by previous observations, clearly violating the independence assumption. Ironically, if the data was really independent, we would not need samples in the first place to estimate parameters because the probabilities would already be known! Given

¹A thumb tack consists of a short pin with a flat circular head and is used to fasten items such as a piece of paper to a wall or bulletin board. A thumb tack is more interesting for these experimental discussions than a coin, since coins typically have a probability of heads equal to roughly 50%, regardless of the specifics of the coin.

this, if you were now presented with a thumb tack from a box that was to be repeatedly tossed, do you think that the tosses are independent?

This example highlights the tenuous foundations, an odd circularity, and the internal inconsistency of the frequentist approach that proceeds under the assumption of a fixed “true” parameter. All frequentist procedures are founded on the assumption of known parameter values: sampling distributions of estimators are computed conditional on θ ; confidence intervals consist of calculations of the form: $P(f(y_1, \dots, y_T) \in (a, b) | \theta)$; and asymptotics also all rely on the assumption of known parameter values. None of these calculations are possible without assuming the known parameters.

In the frequentist approach, even though the parameter is completely unknown to the researcher, θ is not a random variable, does not have a distribution, and therefore inference is not governed by the rules of probability. Given this “fixed, but unknown” definition, it is impossible to discuss concepts like “parameter uncertainty.” This strongly violates our intuition, since things that are not known are typically thought of as random.

The Bayesian approach avoids this internal inconsistency by shedding the strong assumption of independence and assumption of a fixed but unknown parameter. Instead it assumes that θ is a random variable and describes the uncertainty about θ using a probability distribution, $p(\theta)$ (the prior). The joint distribution of the data is then

$$p(y_1, \dots, y_T) = \int p(y_1, \dots, y_T | \theta)p(\theta)d\theta = \int \prod_{t=1}^T p(y_t | \theta)p(\theta)d\theta.$$

Notice, that the right-hand-side does not depend on the order of the data, and the joint distribution of the data is the same for all potential orderings. This is a natural assumption about the symmetry of the data, and is called *exchangeability*. The Bayesian approach makes no assumptions about the order in which the data may arrive, and each observation has the same marginal distribution, $P(y_i = 1) = P(y_j = 1)$ for any i and j .

Thus, we replace the independence assumption with a weaker and more natural assumption of exchangeability: a collection of random variables, y_1, \dots, y_T , is exchangeable if the distribution of y_1, \dots, y_T is the same as the distribution of any permutation $y_{\pi_1}, \dots, y_{\pi_T}$, where $\pi = (\pi_1, \dots, \pi_T)$ is a permutation of the integers 1 to T . Independent events are always exchangeable, but the converse is not true. Notice the differences between the assumptions in the Bayesian and frequentist approach: the Bayesian makes assumptions over potentially realized data, and there is no need to invent the construct of a fixed but unknown parameter, since exchangeability makes no reference to parameters.

In the case of the tack throwing experiment, exchangeability states that the ordering of heads and tails does not matter. Thus, if the experiment of 8 tosses generated 4 heads, it does not matter if the ordering was $(1, 0, 1, 0, 1, 0, 1, 0)$ or

$(0, 1, 1, 0, 1, 0, 0, 1)$. This is a natural assumption about the symmetry of the tack tosses, capturing the idea that the information in any toss or sequence of tosses is the same as any other—the idea of a truly random sample. It is important to note that exchangeability is a property that applies prior to viewing the data. After observation, data is no longer a random variable, but a realization of a random variable.

Bruno de Finetti introduced the notion of exchangeability, and then asked a simple question: “What do exchangeable sequences of random variables look like?” The answer to this question is given in the famous de Finetti’s theorem, which also *defines* models, parameters, and provides important linkages between frequentist and classical statistics.

3.1.1 de Finetti’s representation theorem

de Finetti’s representation theorem provides the theoretical connection between data, models, and parameters. It is stated first in the simplest setting, where the observed data takes two values, either zero or one, and then extended below.

Theorem 3.1 (de Finetti’s representation theorem). *Let (Y_1, Y_2, \dots) be an infinite sequence of 0-1 exchangeable random variables with joint density $p(Y_1, \dots, Y_n)$. Then there exists a distribution P such that*

$$P(Y_1 = y_1, \dots, Y_n = y_n) = \int \prod_{i=1}^n \theta^{y_i} (1 - \theta)^{1-y_i} dP(\theta) = \int \prod_{i=1}^n p(y_i | \theta) dP(\theta) \quad (3.1)$$

where

$$P(\theta) = \lim_{n \rightarrow \infty} \text{Prob} \left[\frac{1}{n} \sum_{t=1}^n Y_t \leq \theta \right] \text{ and } \theta = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{t=1}^n Y_t.$$

If the distribution function or measure admits a density with respect to Lebesgue measure, then $dP(\theta) = p(\theta) d\theta$.

Heath and Sudderth (1976) provide a general version and simple proof. For general spaces \mathcal{Y} , we have

$$P(Y_1 \leq y_1, \dots, Y_n \leq y_n) = \int_{\mathcal{F}} \prod_{i=1}^n F(y_i) \mu(dF)$$

where $\mu(\cdot)$ is a measure over the space of distributions, \mathcal{F} . In the parametric case, F_θ , then $\mu(d\theta) = p(\theta)d\theta$ and $p(\theta)$ is the typical prior density. The predictive $p(Y_{n+1} | y_1, \dots, y_n)$ is then a marginal over F_θ and the posterior $p(\theta | y_1, \dots, y_n)$ which is updated via Bayes theorem.

de Finetti’s representation theorem has profound implications for understanding models from a subjectivist perspective and in relating subjectivist to frequentist theories of inference. The theorem is interpreted as follows:

- Under exchangeability, parameters exist, and one can *act as if* the y_i 's are drawn independently from a Bernoulli distribution with parameter θ . That is, they are draws from the model $p(y_i | \theta) = \theta^{y_i} (1-\theta)^{1-y_i}$, generating a likelihood function $p(y | \theta) = \prod_{i=1}^n p(y_i | \theta)$. Formally, the likelihood function is defined via the density $p(y | \theta)$, viewed as a function of θ for a fixed sample $y = (y_1, \dots, y_n)$. More "likely" parameter values generate higher likelihood values, thus the name. The maximum likelihood estimate (MLE) is

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta \in \Theta} p(y | \theta) = \arg \max_{\theta \in \Theta} \ln p(y | \theta),$$

where Θ is the parameter space.

- Parameters are random variables. The limit $\theta = \lim_{n \rightarrow \infty} n^{-1} \sum_{i=1}^n y_i$ exists but is a random variable. This can be contrasted with the strong law of large numbers that requires independence and implies that $n^{-1} \sum_{i=1}^n y_i$ converges almost surely to a fixed value, θ_0 . From this, one can interpret a parameter θ as a limit of observables and justifies the frequentist interpretation of θ as a limiting frequency of 1's.
- The distribution $P(\theta)$ or density $p(\theta)$ can be interpreted as beliefs about the limiting frequency θ prior to viewing the data. After viewing the data, beliefs are updated via Bayes rule resulting in the posterior distribution, $p(\theta | y_1, \dots, y_n)$.

Since the likelihood function is fixed in this case, any distribution of observed data can be generated by varying the prior distribution.

The main implication of de Finetti's theorem is a complete justification for Bayesian practice of treating the parameters as random variables and specifying a likelihood and parameter distribution. Stated differently, a "model" consists of both a likelihood and a prior distribution over the parameters. Thus, parameters as random variables and priors are a necessity for statistical inference, and not some extraneous component motivated by philosophical concerns.

More general versions of de Finetti's theorem are available. A general version is as follows. If $\{Y_i\}_{i \geq 1}$, $Y_i \in \mathbb{R}$, is a sequence of infinitely exchangeable random variables, then there exists a probability measure P on the space of all distribution functions, such that

$$p(Y_1 \leq y_1, \dots, Y_n \leq y_n) = \int \prod_{i=1}^n F(y_i) P(dF)$$

with mixing measure

$$P(F) = \lim_{n \rightarrow \infty} P(F_n),$$

where F_n is the empirical distribution of the data. At this level of generality, the distribution function is infinite-dimensional. In practice, additional subjective assumptions are needed that usually restrict the distribution function to finite dimensional spaces, which implies that the distribution function is indexed by a parameter vector θ :

$$p(Y_1 \leq y_1, \dots, Y_n \leq y_n) = \int \prod_{i=1}^n p(Y_i | \theta) dP(\theta).$$

To operationalize this result, the researcher needs to choose the likelihood function and the prior distribution of the parameters.

3.1.2 Posterior Empirical CDF

Let $m = \{f_\theta(y) : y \in \mathcal{Y}\}$ be a model. When necessary we index the parameters in model m , as θ_m . Let $y = (y_1, \dots, y_n)$ be a vector of signals. The conditional likelihood, under m , is given by $f_\theta(y) = \prod_{i=1}^n f_\theta(y_i)$. We also allow for the possibility that the data is generated from a model f that does not belong to the family of models f_θ .

Given a prior measure, $\Pi(dF)$, over \mathcal{F} the set of distributions, we can calculate the predictive density

$$f_n(y_{n+1} | y_1, \dots, y_n) = \int f(y) \Pi_n(df) \text{ where } \Pi_n(df) = \frac{\prod_{i=1}^n f(y_i) \Pi(df)}{\int \prod_{i=1}^n f(y_i) \Pi(df)}$$

Under the family, f_θ , we can calculate the parameter posterior as

$$p(\theta | y) = \frac{\prod_{i=1}^n f_\theta(y_i) p(\theta) d\theta}{m(y)} \text{ where } m(y) = \int f_\theta(y) p(\theta) d\theta$$

Here $p(\theta)$ is a prior distribution over parameters and $m(y)$ is the marginal distribution of the data implied by the model. There are many applications in Bayesian non-parametric statistics.

At first glance, de Finetti's theorem may seem to suggest that there is a single model or likelihood function. This is not the case however, as models can be viewed in the same manner as parameters. Denoting a model specification by \mathcal{M} , then de Finetti's theorem would imply that

$$\begin{aligned} p(y_1, \dots, y_n) &= \int \prod_{i=1}^n p(y_i | \theta, \mathcal{M}) p(\theta | \mathcal{M}) p(\mathcal{M}) d\theta d\mathcal{M} \\ &= \int p(y_1, \dots, y_n | \mathcal{M}) p(\mathcal{M}) d\mathcal{M}, \end{aligned}$$

in the case of a continuum of models. Thus, under the mild assumption of exchangeability, it is *as if* the y_i 's are generated from $p(y_i | \theta, \mathcal{M})$, conditional

on the random variables θ and \mathcal{M} , where $p(\theta | \mathcal{M})$ are the beliefs over θ in model \mathcal{M} , and $p(\mathcal{M})$ are the beliefs over model specifications.

Subjective probability is a more general definition of probability than the frequentist definition, as it can be used for all types of events, both repeatable and unrepeatable events. A subjectivist has no issues discussing the probability of a lecture result, even though the underlying conditions has not been observed before. As Ramsey (1926) puts it, “the probability is simply the willingness to bet on an event with a counterpart”.

The event does not even have to be uncertain in nature. For example, the probability of me having coins in my pocket will depend on who is asked to make the assessment. I, knowing the contents of my pocket, will say the probability is 0. However, if you are asked to make the assessment, you will say the probability is 1/2, as you do not know the contents of my pocket. This is a classic example of subjectivist probability.

Similarly, consider the number of people currently in Antarctica. This number is fixed and deterministic at any given moment, yet different individuals will assign different probability distributions to this quantity based on their knowledge. A researcher who recently reviewed Antarctic population statistics might have a tight distribution centered around the correct value, while someone with no such knowledge might have a much wider distribution. A logistics coordinator for a polar research station would have precise information about personnel at their specific facility but uncertainty about other stations. Each of these represents valid subjective probabilities over the same underlying fixed quantity, illustrating how probability in the Bayesian sense quantifies personal uncertainty rather than intrinsic randomness.

The main difficulty in operationalizing subjective probability is the process of actually quantifying subjective beliefs into numeric probabilities. One practical approach is to elicit probabilities through a sequence of carefully designed bets.

Consider eliciting someone’s probability distribution over the number of people in Antarctica. We could start by asking: “Would you accept a bet that pays \$100 if the number is below 5,000, and you pay \$50 if it’s above 5,000?” If they accept, this suggests they believe $P(\text{population} < 5000) > 1/3$. We then adjust the threshold and payoffs systematically. For instance, we might ask about betting on the population being below 2,000, or below 1,000, gradually narrowing down probability mass at different intervals.

For continuous quantities, we can elicit a full distribution through a sequence of binary bets about quantiles. By asking someone to specify values $q_{0.25}, q_{0.5}, q_{0.75}$ such that they are indifferent between bets paying equal amounts if the true value falls below or above each threshold, we construct their 25th, 50th, and 75th percentiles. This process, known as probability elicitation, transforms abstract beliefs into concrete probability distributions by

observing revealed preferences through betting behavior.

The betting framework provides two key advantages. First, it forces coherence: if someone states inconsistent probabilities (such as $P(A) + P(\neg A) \neq 1$), an adversary could construct a Dutch book—a set of bets that guarantees a loss regardless of the outcome. The threat of sure loss incentivizes rational probability assignments. Second, betting naturally handles non-repeatable events. We can elicit probabilities about tomorrow's Supreme Court decision or next quarter's GDP growth, neither of which has a frequentist interpretation.

Instead of using repetitive experiments, subjective probabilities can be measured using betting odds, which have been used for centuries to gauge the uncertainty over an event. The probability attributed to winning a coin toss is revealed by the type of odds one would accept to bet. Notice the difference between the frequentist and Bayesian approach. Instead of defining the probabilities via an infinite repeated experiment, the Bayesian approach elicits probabilities from an individual's observed behavior.

3.2 Sufficient Statistic

In Bayesian inference, we need to compute the posterior over unknown model parameters θ , given data y . The posterior density is denoted by $p(\theta | y)$. A map from data $y = (y_1, \dots, y_n)$ to a statistic $S(y)$ is called a *sufficient statistic* for θ if the conditional distribution of y given $S(y)$ is independent of θ :

$$p(y | S(y), \theta) = p(y | S(y)).$$

This implies that $S(y)$ captures all the information in the data relevant to θ . Because the statistic is a deterministic function of the data, the likelihood factorizes as:

$$p(y | \theta) = p(S(y) | \theta)p(y | S(y)).$$

There is a powerful connection between the posterior mean and sufficient statistics in the exponential family. Kolmogorov (1942) showed that if $S^*(y)$ is a minimal sufficient statistic, then the posterior expectation $E[\theta | y]$ is a function of $S^*(y)$. This provides a theoretical foundation for using summary statistics in simplified models.

Example 3.1 (Posterior Distribution for Coin Toss). What if we gamble against unfair coin flips or the person who performs the flips is trained to get the side he wants? In this case, we need to estimate the probability of heads θ from the data. Suppose we have observed 10 flips

$$\{H, T, H, H, H, T, H, T, H, H\},$$

and only three of them were tails. What is the probability that the next flip will be tail? The frequency-based answer would be $3/10 = 0.3$. However, the Bayes approach gives us more flexibility. Suppose we have a prior belief that the coin is fair, but we are not sure. We model this belief by a prior distribution. Let's discretize the variable θ and assign prior probabilities to each value of θ , the prior distribution is shown in Figure 3.1 (left panel).

```
theta <- seq(0, 1, by = 0.1)
prior <- c(0, 0.024, 0.077, 0.132, 0.173, 0.188, 0.173, 0.132,
         0.077, 0.024, 0)
```

We put most of the mass to the fair assumption ($\theta = 0.5$) and zero mass to the extreme values $\theta = 0$ and $\theta = 1$. Our mass is exponentially decaying as we move away from 0.5. This is a reasonable assumption, since we are not sure about the fairness of the coin. Now, we can use Bayes rule to update our prior belief. The posterior distribution is then combines likelihood shown in Figure 3.1 (middle panel) and prior distributions as shown in Figure 3.1 (right panel).

$$p(\theta | y) = \frac{p(y | \theta)p(\theta)}{p(y)}.$$

The denominator is the marginal likelihood, which is given by

$$p(y) = \sum_{\theta} p(y | \theta)p(\theta).$$

The likelihood is given by the Binomial distribution

$$p(y | \theta) \propto \theta^y (1 - \theta)^{n-y}.$$

Notice, that the posterior distribution depends only on the number of positive and negative cases. Those numbers are *sufficient* for the inference about θ . T

```
par(mar = c(4, 4, 3, 1))
barplot(prior, names.arg = theta, xlab = "theta", ylab =
         "prior", col = "lightblue")

likelihood <- function(theta, n, Y) {
  theta^Y * (1 - theta)^(n - Y)
}
posterior <- likelihood(theta, 10, 3) * prior
posterior <- posterior / sum(posterior) # normalize
barplot(posterior, names.arg = theta, xlab = "theta", ylab =
         "posterior", col = "lightblue")

posterior <- likelihood(theta, 100, 30) * prior
posterior <- posterior / sum(posterior) # normalize
barplot(posterior, names.arg = theta, xlab = "theta", ylab =
         "posterior", col = "lightblue")
```

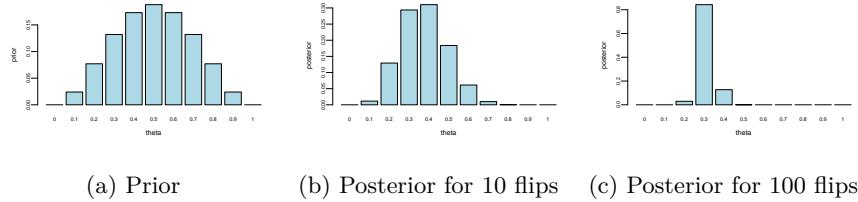


Figure 3.1: Prior and Posterior distribution

If you are to keep collecting more observations and say observe a sequence of 100 flips and 30 of them were heads, then the posterior distribution will be more concentrated around the value of $\theta = 0.3$ as shown in fig-coinposterior (right panel).

This demonstrates that for large sample sizes, the frequentist approach and the Bayesian approach agree.

3.3 Beta-Binomial Model

The *Beta-Binomial Bayesian model* is a statistical model that is used when we are interested in learning about a proportion or probability of success, denoted by p . This model is used, for example, when dealing with binary data in A/B testing.

In the Beta-Binomial model, we assume that the observed data is generated from a Binomial distribution with parameter θ and m trials. The probability of success θ in each of m Bernoulli trials is not fixed but randomly drawn from a Beta distribution. Thus, the model is given by

$$y_i | \theta \sim \text{Binomial}(m, \theta) \text{ (likelihood)}, \quad \theta \sim \text{Beta}(\alpha, \beta) \text{ (prior)}.$$

The *Beta distribution* is a family of continuous probability distributions defined on the interval $[0,1]$ and has two parameters alpha (α) and beta (β), that appear as exponents of the variable and its complement to 1, respectively, and control the shape of the distribution. The Beta distribution is frequently used in Bayesian statistics, empirical Bayes methods, and classical statistics to model random variables with values falling inside a finite interval.

The probability density function (PDF) of the Beta distribution is given by:

$$\text{Beta}(y; \alpha, \beta) = \frac{y^{\alpha-1}(1-y)^{\beta-1}}{B(\alpha, \beta)}$$

where $y \in [0, 1]$, $\alpha > 0$, $\beta > 0$, and $B(\alpha, \beta)$ is the beta function. It is simply a normalizing constant

$$B(\alpha, \beta) = \int_0^1 y^{\alpha-1} (1-y)^{\beta-1} dy.$$

The mean and variance of the Beta distribution are given by:

$$\mu = \frac{\alpha}{\alpha + \beta}$$

$$\sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}$$

where μ is the mean and σ^2 is the variance.

Figure 3.2 illustrates the Beta distribution for different values of α and β .

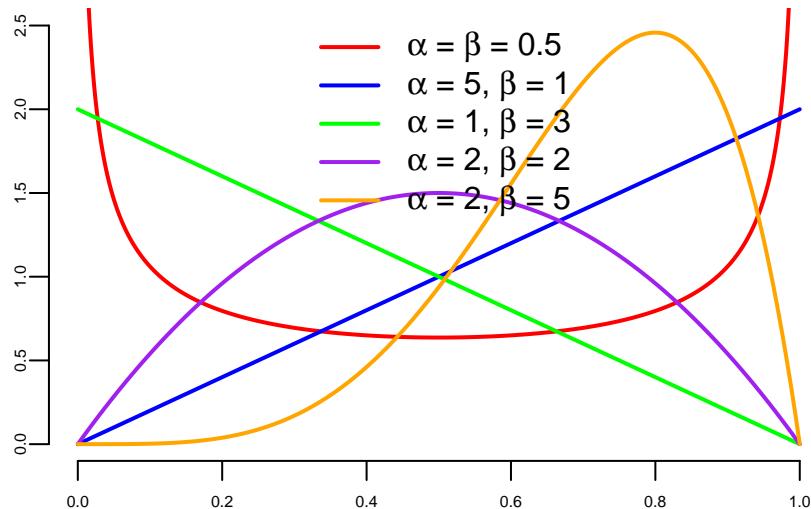


Figure 3.2: Beta distributions

The Beta-Binomial model is one of the simplest Bayesian models and is widely used in various fields including epidemiology, intelligence testing, and marketing. It provides the tools we need to study the proportion of interest, θ , in a variety of settings.

The nice property of the Beta-Binomial model is that the posterior

$$p(\theta | y) = \frac{p(y | \theta)p(\theta)}{p(y)}$$

$p(\theta | y)$ is yet another Beta distribution. Beta is called a *conjugate prior* for the Binomial likelihood and is a very useful property.

Conjugate prior

A prior is called a *conjugate prior* for a likelihood function if the posterior distribution is of the same family as the prior distribution.

When $m = 1$ (observations follow the Bernoulli distribution), the posterior is given by

$$p(\theta | Y) = \text{Beta}(Y + \alpha, 1 - Y + \beta)$$

where Y is the number of successful outcomes

$$Y = \sum_{i=1}^n y_i,$$

where $y_i | \theta \sim \text{Bernoulli}(\theta)$.

Here the count of successful outcome Y acts as a *sufficient statistic* for the parameter θ . This means that the posterior distribution depends on the data only through the sufficient statistic Y . This is a very useful property and is a consequence of the conjugacy of the Beta prior and Binomial likelihood.

In the case of $n > 1$ (observations follow the Binomial distribution), the posterior is given by

$$\theta | Y \sim \text{Beta}(Y + \alpha, n - Y + \beta)$$

where n is the number of observations and Y is the number of successful outcomes as before.

$$Y = \sum_{i=1}^n y_i,$$

where $y_i | \theta \sim \text{Binomial}(n, \theta)$.

The posterior mean and variance are

$$\mathbb{E}[\theta | Y] = \frac{\alpha_n}{\alpha_n + \beta_n} \quad \text{and} \quad \text{Var}(\theta | Y) = \frac{\alpha_n \beta_n}{(\alpha_n + \beta_n)^2 (\alpha_n + \beta_n + 1)},$$

where $\alpha_n = \alpha + Y$ and $\beta_n = \beta + n - Y$.

Example 3.2 (Black Swans). A related problem is the Black Swan inference problem. Suppose that after n trials where n is large you have only seen successes and that you assess the probability of the next trial being a success as $(T+1)/(T+2)$ that is, almost certain. This is a model of observing White Swans and having never seen a Black Swan. Taleb (2007) makes it sound as if the rules of probability are not rich enough to be able to handle Black Swan events. There is a related class of problems in finance known as *Peso problems* where countries decide to devalue their currencies and there is little prior evidence from recent history that such an event is going to happen.

To obtain such a probability assessment we use a Binomial/Beta conjugate Bayes updating model. The key point is that it can also explain that there is still a large probability of a Black Swan event to happen *sometime* in the future. An independence model has difficulty doing this.

The Bayes Learning Beta-Binomial model will have no problem. We model with $y_t = 0$ or 1 , with probability $P(y_t = 1 | \theta) = \theta$. This is the classic Bernoulli “coin-flipping” model and is a component of more general specifications such as regime switching or outlier-type models.

Let $Y = \sum_{t=1}^T y_t$ be the number of observed successful outcomes. The likelihood for a sequence of Bernoulli observations is then

$$p(y | \theta) = \prod_{t=1}^T p(y_t | \theta) = \theta^Y (1 - \theta)^{T-Y}.$$

The maximum likelihood estimator is the sample mean, $\hat{\theta} = T^{-1}Y$. This makes little sense when you just observe white swans. It predicts $\hat{\theta} = 1$ and gets shocked when it sees a black swan (zero probability event). Bayes, on the other hand, allows for “learning”.

Bayes rule then tells us how to combine the likelihood and prior to obtain a posterior distribution, namely $\theta | Y = y$. What do we believe about θ given a sequence of observations? It is straightforward to show that the posterior distribution is again a Beta distribution with

$$p(\theta | y) \sim \text{Beta}(\alpha_n, \beta_n) \text{ and } \alpha_n = \alpha + k, \beta_n = \beta + T - k.$$

Suppose we have observed $T = 100$ white swans and no black swans ($Y = 100$). Using a uniform prior $\text{Beta}(1, 1)$, the posterior is $\text{Beta}(101, 1)$. The predictive probability that the next swan is white is the posterior mean:

$$P(y_{T+1} = 1 | Y = 100) = \mathbb{E}(\theta | Y) = \frac{\alpha_n}{\alpha_n + \beta_n} = \frac{101}{102} \approx 0.99.$$

What about the probability of seeing at least one black swan in the next 100 observations? We compute this by integrating over the posterior:

$$P(\text{at least one black swan in next 100} | Y) = \int_0^1 [1 - \theta^{100}] p(\theta | Y) d\theta.$$

```
## P(next swan is white) = 0.99
## P(at least one black swan in next 100) = 0.5
```

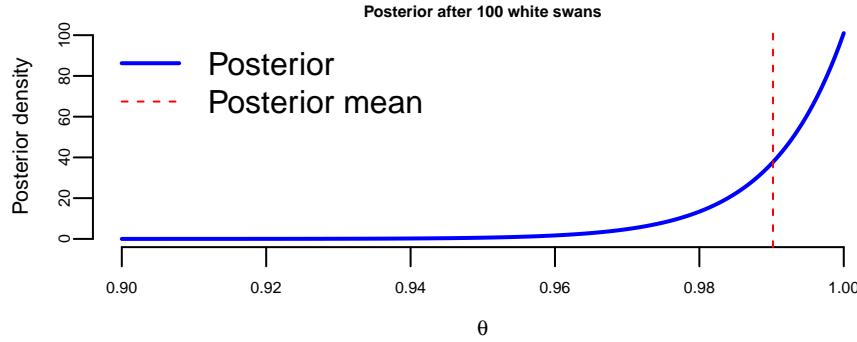


Figure 3.3: Posterior distribution after observing 100 white swans

Despite never having observed a black swan, the Bayesian approach assigns approximately 49.9% probability to seeing at least one black swan in the next 100 trials. This illustrates how Bayesian learning naturally handles rare events by maintaining uncertainty about θ through the posterior distribution.

Example 3.3 (Clinical Trials). Consider a problem of designing clinical trials in which K possible drugs $a \in 1, \dots, K$ need to be tested. The outcome of the treatment with drug a is binary $y(a) \in \{0, 1\}$. We use Bernoulli distribution with mean $f(a)$ to model the outcome. Thus, the full probabilistic model is described by $w = f(1), \dots, f(K)$. Say we have observed a sample $D = \{y_1, \dots, y_n\}$. We would like to compute posterior distribution over w . We start with a Beta prior

$$p(w | \alpha, \beta) = \prod_{a=1}^K \text{Beta}(w_a | \alpha, \beta)$$

Then the posterior distribution is given by

$$p(w | D) = \prod_{a=1}^K \text{Beta}(w_a | \alpha + n_{a,1}, \beta + n_{a,0})$$

This setup allows us to perform sequential design of experiments. The simplest version of it is called Thompson sampling. After observing n patients, we draw a single sample \tilde{w} from the posterior and then maximize the resulting surrogate

$$a_{n+1} = \arg \max_a f_{\tilde{w}}(a), \quad \tilde{w} \sim p(w | D)$$

Example 3.4 (Shrinkage and Baseball Batting Averages). The batter-pitcher match-up is a fundamental element of a baseball game. There are detailed

baseball records that are examined regularly by fans and professionals. This data provides a good illustration of Bayesian hierarchical methods. There is a great deal of prior information concerning the overall ability of a player. However, we only see a small amount of data about a particular batter-pitcher match-up. Given the relatively small sample size, to determine our optimal estimator we build a hierarchical model taking into account the within pitcher variation.

Let's analyze the variability in Jeter's 2006 season. Let p_i denote Jeter's ability against pitcher i and assume that p_i varies across the population of pitchers according to a particular probability distribution $(p_i | \alpha, \beta) \sim Be(\alpha, \beta)$. To account for extra-binomial variation we use a hierarchical model for the observed number of hits y_i of the form

$$(y_i | p_i) \sim Bin(T_i, p_i) \text{ with } p_i \sim Be(\alpha, \beta)$$

where T_i is the number of at-bats against pitcher i . A priori we have a prior mean given by $E(p_i) = \alpha/(\alpha+\beta) = \bar{p}$. The extra heterogeneity leads to a prior variance $Var(p_i) = \bar{p}(1-\bar{p})\phi$ where $\phi = (\alpha + \beta + 1)^{-1}$. Hence ϕ measures how concentrated the beta distribution is around its mean, $\phi = 0$ means highly concentrated and $\phi = 1$ means widely dispersed.

This model assumes that each player i has a true ability p_i that is drawn from a common distribution. The model is hierarchical in the sense that the parameters α and β are estimated from the data. The model is also a shrinkage model in the sense that the estimates of p_i are shrunk towards the overall mean \bar{p}_i . In reality, we don't know that each p_i exists. We also don't know if it follows a Binomial distribution with the Beta prior. We are making a model assumption. However, the model is a good approximation to the data and is a good way to estimate the parameters.

H. Stern et al. (2007) estimates the parameter $\hat{\phi} = 0.002$ for Derek Jeter, showing that his ability varies a bit but not very much across the population of pitchers. The effect of the shrinkage is not surprising. The extremes are shrunk the most with the highest degree of shrinkage occurring for the match-ups that have the smallest sample sizes. The amount of shrinkage is related to the large amount of prior information concerning Jeter's overall batting average. Overall Jeter's performance is extremely consistent across pitchers as seen from his estimates. Jeter had a season .308 average. We see that his Bayes estimates vary from .311 to .327 and that he is very consistent. If all players had a similar record then the assumption of a constant batting average would make sense.

Pitcher	At-bats	Hits	ObsAvg	EstAvg	95% Int
R. Mendoza	6	5	.833	.322	(.282, .394)
H. Nomo	20	12	.600	.326	(.289, .407)

Pitcher	At-bats	Hits	ObsAvg	EstAvg	95% Int
A.J.Burnett	5	3	.600	.320	(.275, .381)
E. Milton	28	14	.500	.324	(.291, .397)
D. Cone	8	4	.500	.320	(.218, .381)
R. Lopez	45	21	.467	.326	(.291, .401)
K. Escobar	39	16	.410	.322	(.281, .386)
J. Wettland	5	2	.400	.318	(.275, .375)
T. Wakefield	81	26	.321	.318	(.279, .364)
P. Martinez	83	21	.253	.312	(.254, .347)
K. Benson	8	2	.250	.317	(.264, .368)
T. Hudson	24	6	.250	.315	(.260, .362)
J. Smoltz	5	1	.200	.314	(.253, .355)
F. Garcia	25	5	.200	.314	(.253, .355)
B. Radke	41	8	.195	.311	(.247, .347)
D. Kolb	5	0	.000	.316	(.258, .363)
J. Julio	13	0	.000	.312	(.243, .350)
Total	6530	2061	.316		

Some major league managers believe strongly in the importance of such data (Tony La Russa, *Three days in August*). One interesting example is the following. On Aug 29, 2006, Kenny Lofton (career .299 average, and current .308 average for 2006 season) was facing the pitcher Milton (current record 1 for 19). He was *rested* and replaced by a .273 hitter. Is putting in a weaker player really a better bet? Was this just an over-reaction to bad luck in the Lofton-Milton match-up? Statistically, from Lofton's record against Milton we have $P(\leq 1 \text{ hit in } 19 \text{ attempts} | p = 0.3) = 0.01$ an unlikely 1-in-100 event. However, we have not taken into account the multiplicity of different batter-pitcher match-ups. We know that Lofton's batting percentage will vary across different pitchers, it's just a question of how much? A hierarchical analysis of Lofton's variability gave a $\phi = 0.008$ – four times larger than Jeter's $\phi = 0.002$. Lofton has batting estimates that vary from .265 to .340 with the lowest being against Milton. Hence, the optimal estimate for a pitch against Milton is $.265 < .275$ and resting Lofton against Milton is justified by this analysis.

3.4 Poisson Model for Count Data

The Poisson distribution is obtained as a result of the Binomial when p is small and n is large. In applications, the Poisson models count data. Suppose we want to model the arrival rate of users to one of our stores. Let $\lambda = np$, which is fixed and take the limit as $n \rightarrow \infty$. There is a relationship between

$p(x)$ and $p(x + 1)$ given by

$$\frac{p(x+1)}{p(x)} = \frac{\binom{n}{x+1} p^{x+1} (1-p)^{n-x-1}}{\binom{n}{x} p^x (1-p)^{n-x}} \approx \frac{np}{x+1}$$

If we approximate $p(x+1) \approx \lambda p(x)/(x+1)$ with $\lambda = np$, then we obtain the Poisson pdf given by $p(x) = p(0)\lambda^x/x!$. To ensure that $\sum_{x=0}^{\infty} p(x) = 1$, we set

$$f(0) = \frac{1}{\sum_{x=0}^{\infty} \lambda^x / x!} = e^{-\lambda}.$$

The above equality follows from the power series property of the exponent function

$$e^{\lambda} = \sum_{x=0}^{\infty} \frac{\lambda^x}{x!}$$

The *Poisson distribution* counts the occurrence of events. Given a rate parameter, denoted by λ , we calculate probabilities as follows

$$p(X = x) = \frac{e^{-\lambda} \lambda^x}{x!} \text{ where } x = 0, 1, 2, 3, \dots$$

For n independent Poisson observations x_1, \dots, x_n , the sufficient statistic for λ is the sum $\sum_{i=1}^n x_i$. The mean and variance of the Poisson are given by:

Poisson Distribution	Parameters
Expected value	$\mu = E(X) = \lambda$
Variance	$\sigma^2 = \text{Var}(X) = \lambda$

Here λ denotes the rate of occurrence of an event.

Consider the problem of modeling soccer scores in the English Premier League (EPL) games. We use data from Betfair, a website, which posts odds on many football games. The goal is to calculate odds for the possible scores in a match.

0 – 0, 1 – 0, 0 – 1, 1 – 1, 2 – 0, ...

Another question we might ask, is what's the odds of a team winning? This is given by $P(X > Y)$. The odds of a draw are given by $P(X = Y)$. Here X is the number of goals scored by the home team and Y is the number of goals scored by the away team.

Professional sports bettors rely on sophisticated statistical models to predict the outcomes. Instead, we present a simple, but useful model for predicting

outcomes of EPL games. We follow the methodology given in Spiegelhalter and Ng (2009).

To make the discussion more concrete, we will use data from the English Premier League (EPL) 2014/2015 season and model the game between Manchester United and Hull City.

First, load the data and then model the number of goals scored using Poisson distribution for each team.

home_team_name	away_team_name	home_score	guest_score
Arsenal	Liverpool	3	4
Bournemouth	Manchester United	1	3
Burnley	Swansea	0	1
Chelsea	West Ham	2	1
Crystal Palace	West Bromwich Albion	0	1
Everton	Tottenham	1	1

Let's compare the empirical distribution across the number of goals scored by Manchester United to the Poisson distribution.

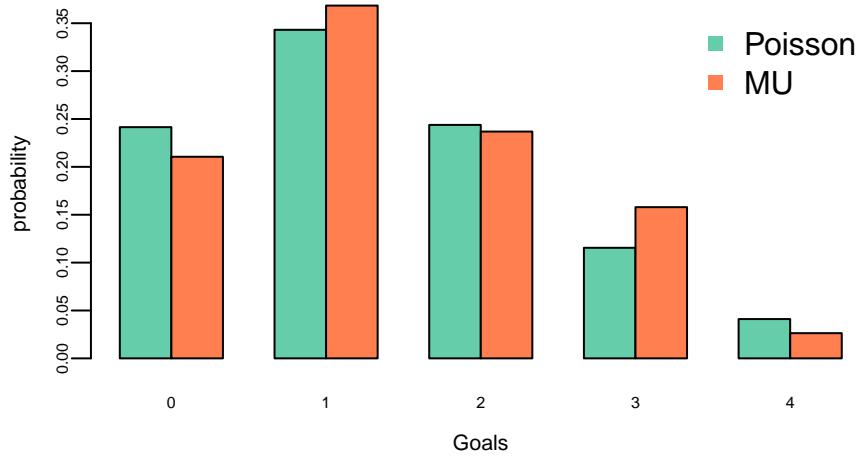


Figure 3.4: Histogram vs Poisson Model Prediction of Goals Scored by MU

Hence the historical data fits closely to a Poisson distribution, the parameter λ describes the average number of goals scored and we calculate it by calculating the sample mean, the maximum likelihood estimate. A Bayesian method where we assume that λ has a Gamma prior is also available. This lets you incorporate outside information into the predictive model.

Now we will use Poisson model and Monte Carlo simulations to predict possible outcomes of the MU vs Hull games. First we estimate the rate parameter for goals by MU `lmb_mu` and goals by Hull `lmb_h`. Each team played a home and away game with every other team, thus 38 total games was played by all teams. We calculate the average by dividing total number of goals scored by the number of games

Team	GF_H	GF_A	GA_H	GA_A
Hull	28	9	35	45
Manchester United	26	28	12	17

Summarizing the data

```
lmb_mu <- (26 + 28) / 38
lmb_h <- (28 + 9) / 38
```

Now we simulate 100 games between the teams

```
x <- rpois(100, lmb_mu)
y <- rpois(100, lmb_h)
knitr::kable(table(x, y))
```

	0	1	2	3	4	5
0	11	5	2	2	0	0
1	16	18	5	4	1	1
2	7	5	2	0	0	0
3	8	2	2	1	0	0
4	2	3	1	1	0	0
5	0	1	0	0	0	0

From our simulation that `sum(x>y)`: 48 number of times MU wins and `sum(x==y)`: 32 there is a draw. The actual outcome was 0-0 (Hull at MU) and 0-1 (Mu at Hull). Thus our model gives a reasonable prediction.

The model can be improved by calculating different averages for home and away games. For example, Hull does much better at home games compared to away games. Further, we can include the characteristics of the opponent team to account for interactions between attack strength (number of scored) and defense weakness of the opponent. Now we modify our value of expected goals for each of the teams by calculating

$$\hat{\lambda} = \lambda \times \text{Defense weakness}$$

Let's model the MU at Hull game. The average away goals for MU $28/19 = 1.4736842$ and the defense weakness of Hull is $35/19 = 1.8421053$, thus the adjusted expected number of goals to be scored by MU is 2.7146814 . Similarly, the adjusted number of goals Hull is expected to score is $28/19 \times 17/19 = 1.3185596$

As a result of the simulation, we obtain the following count matrix of possible outcomes, shown in Figure 3.5.

```
set.seed(1)
x <- rpois(100, 28 / 19 * 35 / 19)
y <- rpois(100, 28 / 19 * 17 / 19)
image(z = table(x, y), x = 0:7, y = 0:5, xlab = "MU Score", ylab
      = "Hull Score")
```

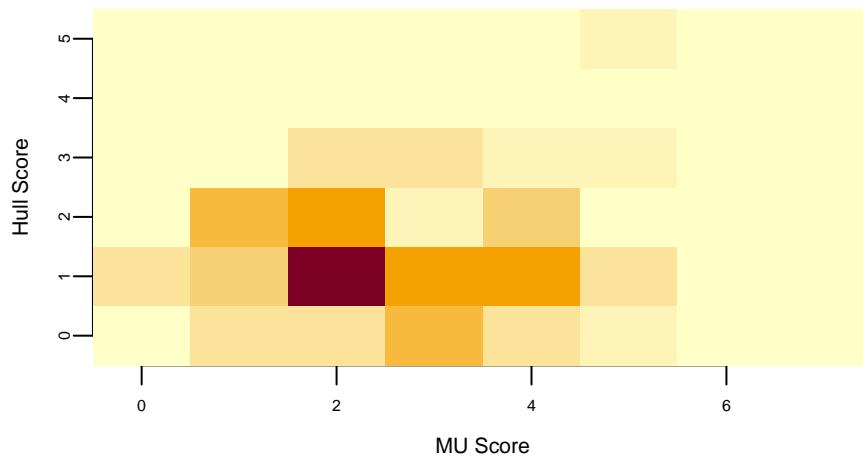


Figure 3.5: Simulation of MU vs Hull game

Now we can calculate the number of times MU wins:

```
sum(x > y)
## 67
```

A model is only as good as its predictions. Let's see how well our model predicted the outcome of the MU vs Hull game. The actual outcome was 0-1 (MU at Hull). The model predicted that most likely MU would score 1-2 (16 games out of 100). In our simulation 0-1 was the third most probable outcome (8 games out of 100). Man U wins 67 games out of 100, we should bet when odds ratio is below 67 to 100.

3.5 Poisson-Gamma: Learning about an Intensity

Consider a continuous-time stochastic process, $\{N_t\}_{t \geq 0}$, with $N_0 = 0$, counting the number of events that have occurred up to time t . The process is constant between event times, and jumps by one at event times: $\Delta N_t = N_t - N_{t-} = 1$, where N_{t-} is the limit from the left. The probability of an event over the next short time interval, Δt is $\lambda \Delta t$, and N_t is called a Poisson process because

$$P[N_t = k] = \frac{e^{-\lambda t} (\lambda t)^k}{k!} \text{ for } k = 1, \dots$$

which is the Poisson distribution, thus $N_t \sim Poi(\lambda t)$. A more general version of the Poisson process is a Cox process, or doubly stochastic point process.

Here, there is additional conditioning information in the form of state variables, $\{X_t\}_{t > 0}$. The process now has two sources of randomness, one associated with the discontinuous jumps and another in the form of random state variables, $\{X_t\}_{t > 0}$, that drive the intensity of the process. The intensity of the Cox process is $\lambda_t = \int_0^t \lambda(X_s) ds$, which is formally defined as

$$P[N_t - N_s = k \mid \{X_u\}_{s \leq u \leq t}] = \frac{\left(\int_s^t \lambda(X_s) ds \right)^k \exp\left(-\int_s^t \lambda(X_s) ds\right)}{k!}, \quad \forall k$$

Cox processes are very useful extensions to Poisson processes and are the basic building blocks of reduced form models of defaultable bonds.

The inference problem is to learn about λ from a continuous-record of observation up to time t . The likelihood function is given by

$$p(N_t = k \mid \lambda) = \frac{(\lambda t)^k \exp(-\lambda t)}{k!},$$

and the MLE is $\hat{\lambda} = N_t/t$. Notice that the total count N_t and elapsed time t together form the sufficient statistic for λ , since the likelihood depends on the data only through these two quantities. The MLE has the unattractive property that prior to the first event $\{t : N_t = 0\}$, the MLE is 0, despite the fact that the model explicitly assumes that events are possible. This problem often arises in credit risk contexts, where it would seem odd to assume that the probability of default is zero just because a default has not yet occurred.

A natural prior for this model is the Gamma distribution, which has the following pdf

$$p(\lambda \mid a, A) = \frac{A^a}{\Gamma(a)} \lambda^{a-1} \exp(-A\lambda). \quad (3.2)$$

Like the beta distribution, a Gamma prior distribution allows for a variety of prior shapes and is parameterized by two hyperparameters. Combining the prior and likelihood, the posterior is also Gamma:

$$p(\lambda | N_t) \propto \frac{(\lambda)^{N_t+a-1} \exp(-\lambda(t+A))}{N_t!} \sim \mathcal{G}(a_t, A_t),$$

where $a_t = N_t + a$ and $A_t = t + A$. The expected intensity, based on information up to time t , is

$$\mathbb{E}[\lambda | N_t] = \frac{a_t}{A_t} = \frac{N_t + a}{t + A} = w_t \frac{N_t}{t} + (1 - w_t) \frac{a}{A},$$

where the second line expresses the posterior mean in shrinkage form as a weighted average of the MLE and the prior mean where $w_t = t/(t + A)$. In large samples, $w_t \rightarrow 1$ and $E(\lambda | N_t) \approx N_t/t = \hat{\lambda}$.

To understand the updating mechanics, Figure 3.6 (right column) displays a simulated sample path, posterior means, and (5%,95%) posterior quantiles for various prior configurations. In this case, time is measured in years and the intensity used to simulate the data is $\lambda = 1$, implying on average one event per year. The four prior configurations embody different beliefs. In the first case, in the middle left panel, $a = 4$ and $A = 1$, captures a high-activity prior, that posits that jumps occur, on average, four times per year, and there is substantial prior uncertainty over the arrival rate as the (5%,95%) prior quantiles are (1.75,6.7). In the second case, captures a prior that is centered over the true value with modest prior uncertainty. The third case captures a low-activity prior, with a prior mean of 0.2 jumps/year. The fourth case captures a dogmatic prior, that posits that jumps occur three times per year, with high confidence in these beliefs.

The priors were chosen to highlight different potential paths for Bayesian learning. The first thing to note from the priors is the discontinuity upward at event times, and the exponential decrease during periods of no events, both of which are generic properties of Bayesian learning in this model. If one thinks of the events as rare, this implies rapid revisions in beliefs at event times and a constant drop in estimates of the intensity in periods of no events. For the high-activity prior and the sample path observed, the posterior begins well above $\lambda = 1$, and slowly decreases, getting close to $\lambda = 1$ at the end of the sample. This can be somewhat contrasted with the low-activity prior, which has drastic revisions upward at jump times. In the dogmatic case, there is little updating at event times. The prior parameters control how rapidly beliefs change, with noticeable differences across the priors.

In all cases, the orange line shows the cumulative event count N_t , the blue dashed line represents the posterior mean of λ , and the grey dashed lines indicate the 5% and 95% posterior quantiles. The discontinuous upward jumps

at event times and exponential decay during quiet periods are characteristic features of Bayesian learning in Poisson processes.

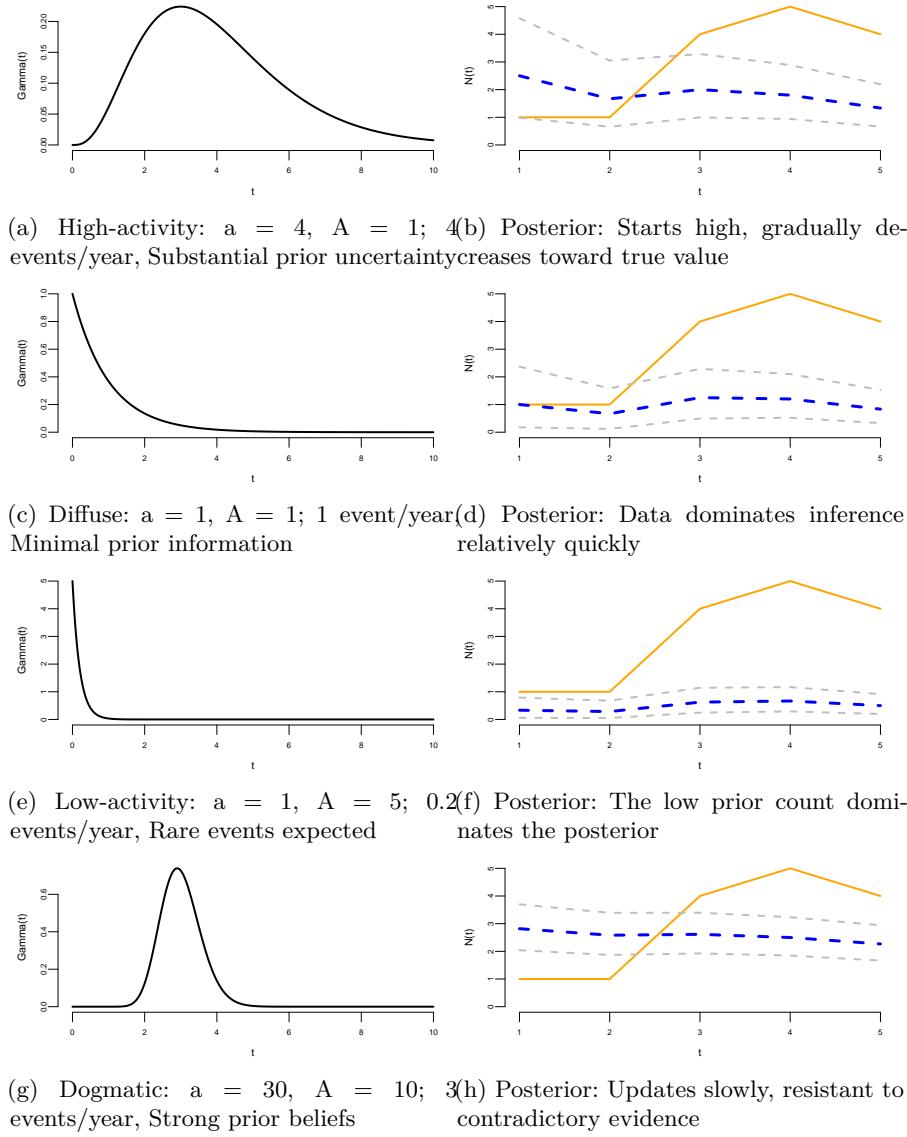


Figure 3.6: Sensitivity of Gamma Prior for Poisson Process

Poisson event models are often embedded as portion of more complicated model to capture rare events such as stock market crashes, volatility surges, currency revaluations, or defaults. In these cases, prior distributions are often

important—even essential—since it is common to build models with events that could, but have not yet occurred. These events are often called ‘Peso’ events. For example, in the case of modeling corporate defaults a researcher wants to allow for a jump to default. This requires positing a prior distribution that places non-zero probability on an event occurring. Classical statistical methods have difficulties dealing with these situations since the MLE of the jump probability is zero, until the first event occurs.

3.6 Exponential-Gamma Model

The Exponential distribution is often used to model waiting times between events, such as the time between independent arrivals in a Poisson process. The probability density function (PDF) is defined as:

$$p(x | \lambda) = \lambda e^{-\lambda x}, \quad x \geq 0$$

where $\lambda > 0$ is the rate parameter (inverse of the mean).

The *Exponential-Gamma* model assumes that the data follows an exponential distribution, and the rate parameter λ follows a Gamma prior distribution.

$$\begin{aligned} \lambda &\sim \text{Gamma}(\alpha, \beta) \\ y_i | \lambda &\sim \text{Exponential}(\lambda) \end{aligned}$$

The probability density function of the Gamma prior is:

$$p(\lambda | \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda}$$

Given n observations $y = (y_1, \dots, y_n)$, the likelihood depends on the data only through the count n and the sum $\sum_{i=1}^n y_i$, which form the sufficient statistics for λ . The posterior distribution of λ is:

$$p(\lambda | y) \propto p(y | \lambda)p(\lambda) \propto \left(\prod_{i=1}^n \lambda e^{-\lambda y_i} \right) \lambda^{\alpha-1} e^{-\beta\lambda} = \lambda^{\alpha+n-1} e^{-(\beta+\sum y_i)\lambda}$$

This is a Gamma distribution with updated parameters:

$$\lambda | y \sim \text{Gamma} \left(\alpha + n, \beta + \sum_{i=1}^n y_i \right).$$

The posterior mean is:

$$\mathbb{E}[\lambda | y] = \frac{\alpha + n}{\beta + \sum y_i}.$$

This model is widely used in reliability engineering (failure rates) and survival analysis, where the rate of events is uncertain and varies across populations.

The Normal or Gaussian distribution is central to probability and statistical inference. Suppose that we are trying to predict tomorrow's return on the S&P500. There's a number of questions that come to mind

1. What is the random variable of interest?
2. How can we describe our uncertainty about tomorrow's outcome?
3. Instead of listing all possible values we'll work with intervals instead. The probability of an interval is defined by the area under the probability density function.

Returns are continuous (as opposed to discrete) random variables. Hence a normal distribution would be appropriate - but on what scale? We will see that on the log-scale a Normal distribution provides a good approximation.

The most widely used model for a continuous random variable is the normal distribution. Standard normal random variable Z has the following properties

The standard Normal has mean 0 and has a variance 1, and is written as

$$Z \sim N(0, 1)$$

Then, we have the probability statements of interest

$$\begin{aligned} P(-1 < Z < 1) &= 0.68 \\ P(-1.96 < Z < 1.96) &= 0.95 \end{aligned}$$

In R, we can find probabilities `pnorm(1.96)`: 0.9750021 and quantiles `qnorm(0.9750)`: 1.959964. The quantile function `qnorm` is the inverse of `pnorm`.

A random variable that follows normal distribution with general mean and variance $X \sim N(\mu, \sigma^2)$, has the following properties

$$\begin{aligned} p(\mu - 2.58\sigma < X < \mu + 2.58\sigma) &= 0.99 \\ p(\mu - 1.96\sigma < X < \mu + 1.96\sigma) &= 0.95. \end{aligned}$$

The chance that X will be within 2.58σ of its mean is 99%, and the chance that it will be within 2σ of its mean is about 95%.

The probability model is written $X \sim N(\mu, \sigma^2)$, where μ is the mean, σ^2 is the variance. This can be transformed to a standardized normal via

$$Z = \frac{X - \mu}{\sigma} \sim N(0, 1).$$

For a Normal distribution, we know that $X \in [\mu - 1.96\sigma, \mu + 1.96\sigma]$ with probability 95%. This is a specific property of the Normal curve (the “Empirical Rule”). For *any* distribution, regardless of shape, we can use **Chebyshev’s Inequality** to bound the probability mass. It states that at least $100(1 - 1/k^2)\%$ of values lie within k standard deviations of the mean:

1. At least 75% probability lies within 2σ ($k = 2$).
2. At least 89% probability lies within 3σ ($k = 3$).
3. At least $100(1 - 1/m^2)\%$ lies within $m \times \sigma$ of the mean μ .

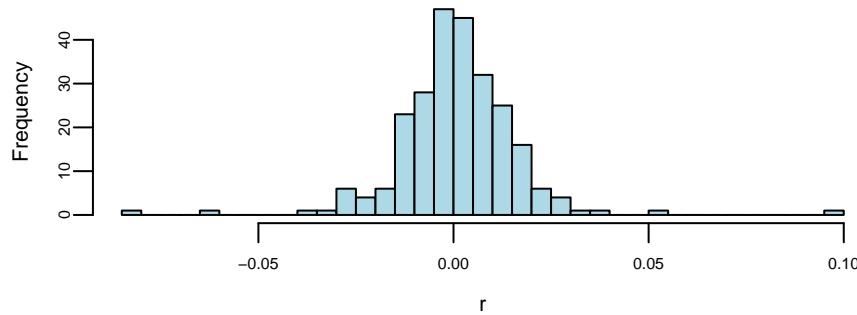
While the Normal distribution guarantees 95% within 2σ , Chebyshev guarantees only 75%, reflecting the uncertainty given a lack of distributional assumptions.

Example 3.5 (Google Stock 2019). Consider observations of daily log-returns of a Google stock for 2019 Daily log-return on day t is calculated by taking a logarithm of the ratio of price at close of day t and at close of day $t - 1$

$$y_t = \log\left(\frac{P_t}{P_{t-1}}\right)$$

For example on January 3 of 2017, the open price is 778.81 and close price was 786.140, then the log-return is $\log(786.140/778.81) = -0.0094$. It was empirically observed that log-returns follow a Normal distribution. This observation is a basis for Black-Scholes model with is used to evaluate future returns of a stock.

```
p <- read.csv("../data/GOOG2019.csv")$Adj.Close
n <- length(p)
r <- log(p[2:n] / p[1:(n - 1)])
hist(r, breaks = 30, col = "lightblue", main = "")
```



Observations on the far right correspond to the days when positive news was released and on the far left correspond to bad news. Typically, those are days when the quarterly earnings reports are released.

To estimate the expected value μ (return) and standard deviation σ (a measure of risk), we simply calculate their sample counterparts

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \text{ and } s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

The empirical (or sample) values \bar{x} and s^2 are called sample mean and sample variance. Here simply view them as our best guess about the mean and variance of the normal distribution model then our probabilistic model for next day's return is then given by

$$R \sim N(\bar{x}, s^2).$$

Say we are interested in investing into Google and would like to calculate the expected return of our investment as well as risk associated with this investment. We assume that behavior of the returns in the future will be the same as in 2019.

```
n <- length(r)
rbar <- sum(r) / n
print(rbar)
## 0.00098
s2 <- sum((r - rbar)^2) / (n - 1)
print(s2)
## 0.00023
x <- seq(-0.08, 0.08, length.out = 200)
hist(r, breaks = 30, col = "lightblue", freq = F, main = "",
     xlab = "")
lines(x, dnorm(x, rbar, sqrt(s2)), col = "red", lwd = 3)
```

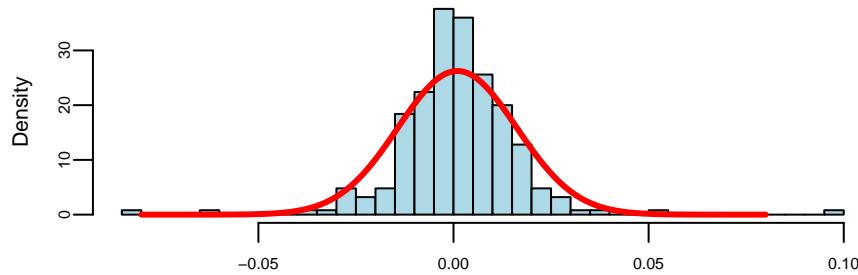


Figure 3.7: Histogram (blue) and fitted normal curve (red) of for the Google stock daily return data.

Now, assume, I invest all my portfolio into Google. I can predict my annual return to be $251 \times 0.0009798 = 0.2459348$ and risk (volatility) of my investment is $\sqrt{s^2} = 1.5198424\%$ a year.

I can predict the risk of losing 3% or more in one day using my model is 1.93%.

```
pnorm(log(1 - 0.03), rbar, sqrt(s2)) * 100
## 1.9
```

```
mean(spret)
## 0.012
sd(spret)
## 0.043
```

3.7 Normal With Unknown Mean

Let Y be a random variable with a normal distribution, $Y \sim N(\mu, \sigma^2)$. The mean μ is unknown, but the variance σ^2 is known. The likelihood function is given by

$$p(y | \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - \mu)^2\right)$$

The MLE of μ is $\hat{\mu} = \bar{y}$, the sample mean. Normal prior for the mean parameter μ is conjugate to the normal likelihood.

$$\mu \sim N(\mu_0, \sigma_0^2)$$

The posterior distribution is also normal.

$$p(\mu | y) \sim N(\mu_n, \sigma_n^2)$$

where

$$\mu_n = \frac{\sigma^2}{n\sigma_0^2 + \sigma^2}\mu_0 + \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}\bar{y}$$

and

$$\sigma_n^2 = \frac{\sigma^2\sigma_0^2}{n\sigma_0^2 + \sigma^2}$$

The posterior mean is a weighted average of the prior mean and the sample mean, with the weights being proportional to the precision of the prior and the likelihood. The posterior variance is smaller than the prior variance, and the sample size n appears in the denominator. The posterior mean is a shrinkage

estimator of the sample mean, and the amount of shrinkage is controlled by the prior variance σ_0^2 . A couple of observations

$$\frac{\sigma^2}{n\sigma_0^2 + \sigma^2} \rightarrow 0 \text{ and } \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2} \rightarrow 1, \text{ as } n \rightarrow \infty.$$

Further,

$$\frac{\sigma^2\sigma_0^2}{n\sigma_0^2 + \sigma^2} \rightarrow 0 \text{ as } n \rightarrow \infty.$$

Example 3.6 (Stylized Example). Assuming the prior distribution $\mu \sim N(-1, 1)$, say we observed $y = 2$ and we want to update our beliefs about μ . The likelihood function is $p(y | \mu) = N(\mu, 2)$, and the posterior distribution is

$$p(\mu | y) \propto p(y | \mu)p(\mu) = N(y | \mu, 2)N(\mu | -1, 1) = N(-0.4, 0.9).$$

```
## "Posterior mean: -0.400000, Posterior variance: 0.894427"
```

Graphically we can represent this as follows

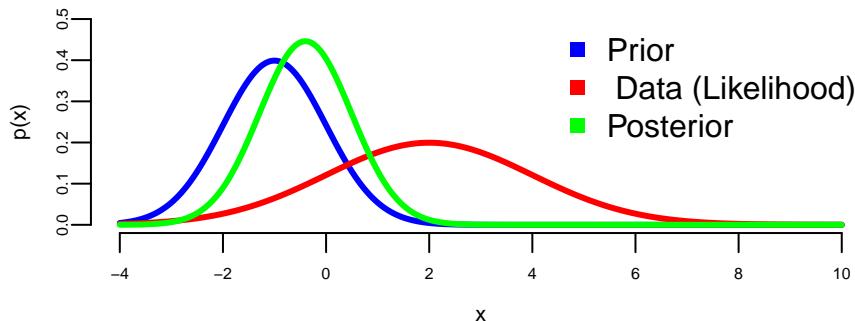


Figure 3.8: Norm-Norm Updating

Note, the posterior mean is in between those of prior and likelihood and posterior variance is lower than variance of both prior and likelihood, this is effect of combining information from data and prior!

More generally, when we observe n independent and identically distributed (i.i.d.) data points y_1, \dots, y_n from a normal distribution with known variance σ^2 , the likelihood function is given by

$$p(y | \mu) = N(\bar{y} | \mu, \sigma^2/n), \text{ where } \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i.$$

Note, that average over the observed data $\bar{y} = \text{Ave}(y_1, \dots, y_n)$ is the sufficient statistics for the mean μ . The prior distribution is given by

$$p(\mu) = N(\mu | \mu_0, \sigma_0^2)$$

The posterior distribution is given by

$$\begin{aligned} p(\mu | y) &\propto \exp \left[\frac{-\mu^2 + 2\mu\mu_0 - \mu_0^2}{2\sigma_0^2} \right] \exp \left[\frac{-\mu^2 + 2\mu\bar{y} - \bar{y}^2}{2\sigma^2/n} \right] \\ &\propto \exp \left[\frac{-\mu^2 + 2\mu\mu_0}{2\sigma_0^2} \right] \exp \left[\frac{-\mu^2 + 2\mu\bar{y}}{2\sigma^2/n} \right]. \end{aligned}$$

Now we combine the terms

$$p(\mu | y) \propto \exp \left[\frac{(-\mu^2 + 2\mu\mu_0)\sigma^2 + (-\mu^2 + 2\mu\bar{y})n\sigma_0^2}{2\sigma_0^2\sigma^2} \right].$$

Now re-arrange and combine μ^2 and μ terms

$$\begin{aligned} p(\mu | y) &\propto \exp \left[\frac{-\mu^2(n\sigma_0^2 + \sigma^2) + 2\mu(\mu_0\sigma^2 + \bar{y}n\sigma_0^2)}{2\sigma_0^2\sigma^2} \right] \\ &\propto \exp \left[\frac{-\mu^2 + 2\mu \left(\frac{\mu_0\sigma^2 + \bar{y}n\sigma_0^2}{n\sigma_0^2 + \sigma^2} \right)}{2(\sigma_0^2\sigma^2)/(n\sigma_0^2 + \sigma^2)} \right]. \end{aligned}$$

Now we add constants which do not depend upon μ to complete the square in the numerator:

$$p(\mu | y) \propto \exp \left[\frac{-\left(\mu - \frac{\mu_0\sigma^2 + \bar{y}n\sigma_0^2}{n\sigma_0^2 + \sigma^2} \right)^2}{2(\sigma_0^2\sigma^2)/(n\sigma_0^2 + \sigma^2)} \right].$$

Finally we get the posterior mean

$$\mu_n = \frac{\mu_0\sigma^2 + \bar{y}n\sigma_0^2}{n\sigma_0^2 + \sigma^2} = \mu_0 \frac{\sigma^2}{n\sigma_0^2 + \sigma^2} + \bar{y} \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}$$

and the posterior variance

$$\sigma_n^2 = \frac{\sigma_0^2\sigma^2}{n\sigma_0^2 + \sigma^2}.$$

Example 3.7 (Chicago Bears 2014-2015 Season). The Chicago Bears are a professional American football team based in Chicago, Illinois. The Bears were a young team in 2014-2015, and were last in their division. This season the Chicago Bears suffered back-to-back 50-points defeats and lost to Patriots and Packers.

- Patriots-Bears 51 – 23
- Packers-Bears 55 – 14

Their next game was at home against the Minnesota Vikings. Current line against the Vikings was -3.5 points. Slightly over a field goal. What's the Bayes approach to learning the line? We use hierarchical data and Bayes learning to update our beliefs in light of new information. The current average win/lose this year can be modeled as a normal distribution with mean μ and standard deviation σ . We assume that μ is normally distributed with mean μ_0 and standard deviation τ .

$$\bar{y} \mid \mu \sim N\left(\mu, \frac{\sigma^2}{n}\right) \sim N\left(\mu, \frac{18.34^2}{9}\right)$$

$$\mu \sim N(0, \tau^2)$$

Here $n = 9$ games so far. With $s = 18.34$ points. We assume the pre-season prior mean $\mu_0 = 0$, standard deviation $\tau = 4$. Based on the observed data so-far: $\bar{y} = -9.22$.

The Bayes Shrinkage estimator is then

$$\mathbb{E}(\mu \mid \tau, \bar{y}) = \frac{\tau^2}{\tau^2 + \frac{\sigma^2}{n}} \bar{y}.$$

The shrinkage factor is 0.3! That's quite a bit of shrinkage. Why? Our updated estimator is

$$\mathbb{E}(\mu \mid \bar{y}, \tau) = -2.75 > -3.5$$

where current line is -3.5 .

- Based on our hierarchical model this is an over-reaction. One point change on the line is about 3% on a probability scale.
- Alternatively, calculate a market-based τ given line = -3.5 .

$$\tau^2 = \frac{\sigma^2}{n} \frac{1}{0.3^2} = 18.34^2 \frac{1}{0.3^2} = 180.$$

- The market-based τ is 13.4 points.

```
bears <- c(-3, 8, 8, -21, -7, 14, -13, -28, -41)
print(mean(bears))
## -9.2
print(sd(bears))
## 18
tau <- 4
sig2 <- sd(bears) * sd(bears) / 9
```

```

print(tau^2 / (sig2 + tau^2))
## 0.3
print(0.29997 * -9.22)
## -2.8
print(pnorm(-2.76 / 18))
## 0.44

```

Home advantage is worth 3 points. The actual result of the game is Bears 21, Vikings 13.

3.7.1 Posterior Predictive

After estimating the parameters, using the posterior distribution, we often want to predict future observations. This is done using the posterior predictive distribution. The posterior predictive distribution is the distribution of a new observation y_{n+1} given the observed data y_1, \dots, y_n . The posterior predictive distribution is given by

$$\begin{aligned}
p(y_{n+1} | y_1, \dots, y_n) &= \int p(y_{n+1} | \mu)p(\mu | y_1, \dots, y_n)d\mu \\
&= \int N(y_{n+1} | \mu, \sigma^2)N(\mu | \mu_n, \sigma_n^2)d\mu = N(y_{n+1} | \mu_n, \sigma_n^2 + \sigma^2).
\end{aligned}$$

This follows from the general properties of the Gaussian distribution

3.8 Normal With Unknown Variance

Consider, another example, when mean μ is fixed and variance is a random variable which follows some distribution $\sigma^2 \sim p(\sigma^2)$. Given an observed sample y , we can update the distribution over variance using the Bayes rule

$$p(\sigma^2 | y) = \frac{p(y | \sigma^2)p(\sigma^2)}{p(y)}.$$

Now, the total probability in the denominator can be calculated as

$$p(y) = \int p(y | \sigma^2)p(\sigma^2)d\sigma^2.$$

A conjugate prior that leads to analytically calculable integral for variance under the normal likelihood is the inverse Gamma. When the mean μ is known, the sufficient statistic for σ^2 is the sum of squared deviations $\sum_{i=1}^n (y_i - \mu)^2$. Thus, if

$$\sigma^2 | \alpha, \beta \sim IG(\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} \sigma^{2(-\alpha-1)} \exp\left(-\frac{\beta}{\sigma^2}\right)$$

and

$$y | \mu, \sigma^2 \sim N(\mu, \sigma^2)$$

Then the posterior distribution is another inverse Gamma $IG(\alpha_{\text{posterior}}, \beta_{\text{posterior}})$, with

$$\alpha_{\text{posterior}} = \alpha + \frac{1}{2}, \quad \beta_{\text{posterior}} = \beta + \frac{y - \mu}{2}.$$

Now, the predictive distribution over y can be calculated by

$$p(y_{\text{new}} | y) = \int p(y_{\text{new}}, \sigma^2 | y) p(\sigma^2 | y) d\sigma^2.$$

Which happens to be a t -distribution with $2\alpha_{\text{posterior}}$ degrees of freedom, mean μ and variance $\alpha_{\text{posterior}}/\beta_{\text{posterior}}$.

3.8.1 The Normal-Gamma Model

Now, consider the case when both mean and variance are unknown. To simplify the formulas, we work with precision $\tau = 1/\sigma^2$. The Normal-Gamma distribution is a conjugate prior for a Normal likelihood with unknown mean and precision. Given data $y = \{y_1, \dots, y_n\}$, we assume:

$$y_i | \theta, \tau \sim N(\theta, 1/\tau)$$

For this model, the sufficient statistics are the sample mean $\bar{y} = n^{-1} \sum_{i=1}^n y_i$, the sample size n , and the sum of squared deviations $\sum_{i=1}^n (y_i - \bar{y})^2$. The Normal-Gamma prior distribution is defined as:

$$\theta | \mu, \tau, \nu \sim N(\mu, 1/(\tau\nu)), \quad \tau | \alpha, \beta \sim \text{Gamma}(\alpha, \beta).$$

Conditional on precision τ , the mean θ is Normal with precision $\nu\tau$. The marginal distribution of τ is Gamma. Note that θ and τ are not independent in the joint prior.

Given the likelihood:

$$p(y | \theta, \tau) \propto \tau^{n/2} \exp\left(-\frac{\tau}{2} \sum_{i=1}^n (y_i - \theta)^2\right)$$

and the prior, the posterior distribution is also Normal-Gamma with parameters:

$$\begin{aligned}\mu_n &= \frac{\nu\mu + n\bar{y}}{\nu + n}, \\ \nu_n &= \nu + n, \\ \alpha_n &= \alpha + \frac{n}{2}, \\ \beta_n &= \beta + \frac{1}{2} \sum_{i=1}^n (y_i - \bar{y})^2 + \frac{n\nu}{2(\nu + n)} (\bar{y} - \mu)^2.\end{aligned}$$

where $\bar{y} = n^{-1} \sum_{i=1}^n y_i$ is the sample mean and n is the sample size. The posterior distribution is a normal-Gamma distribution with parameters $\mu_n, \nu_n, \alpha_n, \beta_n$.

3.8.2 Credible Intervals for Normal-Gamma Model Posterior Parameters

The precision posterior follows a Gamma distribution with parameters α_n, β_n , thus we can use quantiles of the Gamma distribution to calculate credible intervals. A symmetric $100(1-c)$ credible interval $[g_{c/2}, g_{1-c/2}]$ is given by $c/2$ and $1 - c/2$ quantiles of the gamma distribution. To find credible interval for the variance $v = 1/\tau$, we simply use

$$[1/g_{1-c/2}, 1/g_{c/2}].$$

and for standard deviation $s = \sqrt{v}$ we use

$$[\sqrt{1/g_{1-c/2}}, \sqrt{1/g_{c/2}}].$$

To find credible interval over the mean θ , we need to integrate out the precision τ^{-2} from the posterior distribution. The marginal distribution of θ is a Student's t-distribution with parameters center at μ_n , variance $\beta_n/(\nu_n \alpha_n)$ and degrees of freedom $2\alpha_n$.

3.9 Multivariate Normal

We write $X \sim N(\mu, \Sigma)$ for a d -dimensional multivariate normal random vector with mean vector $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$, where Σ is symmetric and positive definite. If the linear algebra below is unfamiliar (transpose,

inverse, determinant, positive definiteness), see Appendix Chapter 26. Its density is

$$p(x | \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu)\right), \quad x \in \mathbb{R}^d.$$

The multivariate normal is the workhorse distribution for joint modeling, conditioning, and linear transformations.

In the multivariate case, the normal-normal model is

$$\theta \sim N(\mu_0, \Sigma_0), \quad y | \theta \sim N(\theta, \Sigma).$$

For a single multivariate observation, y itself serves as the sufficient statistic for θ . More generally, for n independent observations y_1, \dots, y_n , the sample mean $\bar{y} = n^{-1} \sum_{i=1}^n y_i$ is the sufficient statistic. The posterior distribution is

$$\theta | y \sim N(\mu_1, \Sigma_1),$$

where

$$\Sigma_1 = (\Sigma_0^{-1} + \Sigma^{-1})^{-1}, \quad \mu_1 = \Sigma_1(\Sigma_0^{-1}\mu_0 + \Sigma^{-1}y).$$

The predictive distribution is

$$y_{new} | y \sim N(\mu_1, \Sigma_1 + \Sigma).$$

Example 3.8 (Satya Nadella: CEO of Microsoft). In 2014, Satya Nadella became the CEO of Microsoft. The stock price of Microsoft has been on a steady rise since then. Suppose that you are a portfolio manager and you are interested in analyzing the returns of Microsoft stock compared to the market.

Suppose you are managing a portfolio with two positions stock of Microsoft (MSFT) and an index fund that follows S&P500 index and tracks overall market performance. We are interested in estimating the mean returns of the positions in our portfolio. You believe that the returns are normally distributed and are related to each other. You have prior beliefs about these returns, which are also normally distributed. We will use what is called the empirical prior for the mean returns. This is a prior that is based on historical data. The empirical prior is a good choice when you have a lot of historical data and you believe that the future mean returns will be similar to the historical mean returns. We assume the prior for the mean returns is a bivariate normal distribution, let $\mu_0 = (\mu_M, \mu_S)$ represent the prior mean returns for the stocks. The covariance matrix Σ_0 captures your beliefs about the variability and the relationship between these stocks' returns in the prior. We will use the sample mean and covariance matrix of the historical returns as the prior mean and covariance matrix. The prior covariance matrix is given by

$$\Sigma_0 = \begin{bmatrix} \sigma_M^2 & \sigma_{MS} \\ \sigma_{MS} & \sigma_S^2 \end{bmatrix},$$

where σ_M^2 and σ_S^2 are the sample variances of the historical returns of MSFT and SPY, respectively, and σ_{MS} is the sample covariance of the historical returns of MSFT and SPY. The prior mean is given by

$$\mu_0 = \begin{bmatrix} \mu_M \\ \mu_S \end{bmatrix},$$

where μ_M and μ_S are the sample means of the historical returns of MSFT and SPY, respectively. The likelihood of observing the data, given the mean returns, is also a bivariate normal distribution. The mean of this distribution is the true (but unknown) mean returns $\mu = [\mu_A, \mu_B]$. The covariance matrix Σ of the likelihood represents the uncertainty in your data. We will use the sample mean and covariance matrix of the observed returns as the likelihood mean and covariance matrix. The likelihood covariance matrix is given by

$$\Sigma = \begin{bmatrix} \sigma_M^2 & \sigma_{MS} \\ \sigma_{MS} & \sigma_S^2 \end{bmatrix},$$

where σ_M^2 and σ_S^2 are the sample variances of the observed returns of MSFT and SPY, respectively, and σ_{MS} is the sample covariance of the observed returns of MSFT and SPY. The likelihood mean is given by

$$\mu = \begin{bmatrix} \mu_M \\ \mu_S \end{bmatrix},$$

where μ_M and μ_S are the sample means of the observed returns of MSFT and SPY, respectively. In a Bayesian framework, you update your beliefs (prior) about the mean returns using the observed data (likelihood). The posterior distribution, which combines your prior beliefs and the new information from the data, is also a bivariate normal distribution. The mean μ_{post} and covariance Σ_{post} of the posterior are calculated using Bayesian updating formulas, which involve μ_0 , Σ_0 , μ , and Σ .

We use observed returns prior to Nadella's becoming CEO as our prior and analyze the returns post 2014. Thus, our observed data includes July 2015 - Dec 2023 period. We assume the likelihood of observing this data, given the mean returns, is also a bivariate normal distribution. The mean of this distribution is the true (but unknown) mean returns. The covariance matrix *Sigma* of the likelihood represents the uncertainty in your data and is calculated from the overall observed returns data 2001-2023.

```
getSymbols(c("MSFT", "SPY"), from = "2001-01-01", to =
  "2023-12-31")
## "MSFT" "SPY"
s <- 3666 # 2015-07-30
prior <- 1:s
obs <- s:nrow(MSFT) # post covid
```

```
# obs = 5476:nrow(MSFT) # 2022-10-06 bull run if 22-23
a <- as.numeric(dailyReturn(MSFT))
c <- as.numeric(dailyReturn(SPY))
# Prior
mu0 <- c(mean(a[prior]), mean(c[prior]))
Sigma0 <- cov(data.frame(a = a[prior], c = c[prior]))
# Data
mu <- c(mean(a[obs]), mean(c[obs]))
Sigma <- cov(data.frame(a = a, c = c))
# Posterior
SigmaPost <- solve(solve(Sigma0) + solve(Sigma))
muPost <- SigmaPost %*% (solve(Sigma0) %*% mu0 + solve(Sigma)
                           %*% mu)
```

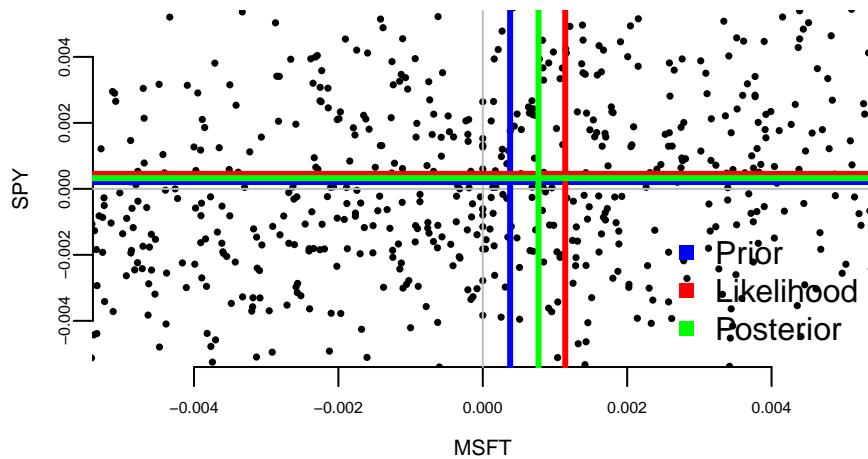


Figure 3.9: Plotting Portfolio Updating with MSFT and SPY

We can see the posterior mean for SPY is close to the prior mean, while the posterior mean for MSFT is further away. The performance of MSFT was significantly better past 2015 compared to SPY. The posterior mean (green) represents mean reversion value. We can think of it a expected mean return if the performance of MSFT starts reverting to its historical averages.

This model is particularly powerful because it can be extended to more dimensions (more stocks) and can include more complex relationships between the variables. It's often used in finance, econometrics, and other fields where understanding the joint behavior of multiple normally-distributed variables is important.

3.10 Mixtures of Conjugate Priors

The mixture of conjugate priors is a powerful tool for modeling complex data. It allows us to combine multiple conjugate priors to create a more flexible model that can capture a wider range of data patterns. The mixture of conjugate priors is particularly useful when the data is generated from a mixture of distributions, where each component of the mixture is generated from a different distribution.

If $p_1(x), \dots, p_k(x)$ are proper density functions and π_1, \dots, π_k are non-negative weights that sum to 1, then the mixture distribution is given by

$$p(x) = \sum_{i=1}^k \pi_i p_i(x).$$

It is easy to show that $p(x)$ is a proper density. Indeed, given domain $x \in A \subset \mathbb{R}$ we have

$$\int_A p(x) dx = \sum_{i=1}^k \pi_i \int_A p_i(x) dx = \sum_{i=1}^k \pi_i = 1.$$

Assume our prior is a mixture of distributions, that is

$$\theta \sim p(\theta) = \sum_{k=1}^K \pi_k p_k(\theta).$$

Then the posterior is also a mixture of normal distributions, that is

$$p(\theta | y) = p(y | \theta) \sum_{k=1}^K \pi_k p_k(\theta) / Z.$$

We introduce a normalizing constant for each component

$$Z_k = \int p(y | \theta) p_k(\theta) d\theta.$$

then

$$p_k(\theta | y) = p_k(\theta) p(y | \theta) / Z_k$$

is a proper distribution and our posterior is a mixture of these distributions

$$p(\theta | y) = \sum_{k=1}^K \pi_k Z_k p_k(\theta | y) / Z.$$

Meaning that we need to require

$$\frac{\sum_{k=1}^K \pi_k Z_k}{Z} = 1, \quad \text{or} \quad Z = \sum_{k=1}^K \pi_k Z_k.$$

Then the posterior density is a mixture

$$p(\theta | y) = \sum_{k=1}^K \hat{\pi}_k p_k(\theta | y).$$

Consider an example of a mixture of two normal distributions. The prior distribution is a mixture of two normal distributions, that is

$$\mu \sim 0.5N(0, 1) + 0.5N(5, 1).$$

The likelihood is a normal distribution with mean μ and variance 1, that is

$$y | \mu \sim N(\mu, 1).$$

The posterior distribution is a mixture of two normal distributions, that is

$$p(\mu | y) \propto \phi(y | \mu, 1) (0.5\phi(\mu | 0, 1) + 0.5\phi(\mu | 5, 1)),$$

where $\phi(x | \mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2 . We can calculate it using property of a normal distribution

$$\phi(x | \mu_1, \sigma_1^2) \phi(x | \mu_2, \sigma_2^2) = \phi(x | \mu_3, \sigma_3^2) \phi(\mu_1 - \mu_2 | 0, \sigma_1^2 + \sigma_2^2)$$

where

$$\mu_3 = \frac{\mu_1/\sigma_1^2 + \mu_2/\sigma_2^2}{1/\sigma_1^2 + 1/\sigma_2^2}, \quad \sigma_3^2 = \frac{1}{1/\sigma_1^2 + 1/\sigma_2^2}.$$

Given, we observed $y = 2$, we can calculate the posterior distribution for μ

```
mu0 <- c(0, 5)
sigma02 <- c(1, 1)
pi <- c(0.5, 0.5)
y <- 2
mu3 <- (mu0 / sigma02 + y) / (1 / sigma02 + 1)
sigma3 <- 1 / (1 / sigma02 + 1)
Z <- dnorm(y - mu0, 0, 1 + sigma02) * pi
w <- Z / sum(Z)
# To add a new line in sprintf, use "\n" inside the format
# string.
print("Component parameters:")
## "Component parameters:"
sprintf(
  "Mean = (%.1f,%.2f)  Var = (%.1f,%.1f)  weights =
  (%.1f,%.2f)",
  mu3[1], mu3[2], sigma3[1], sigma3[2], w[1], w[2]
)
## "Mean = (1.0,3.5)  Var = (0.5,0.5)  weights = (0.65,0.35)"
```

Summary table of random variables

Table 3.6: Summary table of commonly used random variables

Name	θ	PDF	Mean	Variance	Support
Normal	μ, σ^2	$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	μ	σ^2	$x \in \mathbb{R}$
Exponential	λ	$\lambda e^{-\lambda x}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$	$x \geq 0$
Gamma	α, β	$\frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$	$\frac{\alpha}{\beta}$	$\frac{\alpha}{\beta^2}$	$x \geq 0$
Poisson	λ	$\frac{e^{-\lambda} \lambda^x}{x!}$	λ	λ	$x \in \mathbb{N}$
Binomial	n, p	$\binom{n}{x} p^x (1-p)^{n-x}$	np	$np(1-p)$	$x \in \{0, 1, \dots, n\}$
Bernoulli	p	$p^x (1-p)^{1-x}$	p	$p(1-p)$	$x \in \{0, 1\}$
Multinomial	n, p	$\frac{n!}{x_1!x_2!\cdots x_k!} p_1^{x_1} p_2^{x_2} \cdots p_k^{x_k} np_i$	$np_i(1 - \sum_{j \neq i} x_j)$	np_i	$n, x_i \in \mathbb{R}^+$
Beta	α, β	$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1}$	$\frac{\alpha}{\alpha+\beta}$	$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$	$x \in [0, 1]$
Inverse Gamma	α, β	$\frac{\beta^\alpha}{\Gamma(\alpha)} x^{-\alpha-1} e^{-\frac{\beta}{x}}$	$\frac{\beta}{\alpha-1}$	$\frac{\beta^2}{(\alpha-1)^2(\alpha-2)}$	$x > 0$

Table 3.7 summarizes the conjugate prior distributions for common likelihoods. Thus far, we've considered the Normal-Normal model with both known and unknown variance as well as Poisson-Gamma and Beta Binomial. The other pairs are left as an exercise. Given observed data $x = (x_1, \dots, x_n)$ and $s = \sum_{i=1}^n x_i$, $\bar{x} = s/n$. For each conjugate pair, the table lists the sufficient statistics that summarize all information in the data relevant to the parameter. The posterior depends on the data only through these sufficient statistics, which is a key property of conjugate families.

Table 3.7: Conjugate prior table for common likelihoods

Likelihood	Model				
	Prior	Prior Parameters	Pa- rameters	Sufficient Statistics	Posterior Parameters
Normal (known σ^2)	Normal	μ_0, σ_0^2	μ	\bar{x}, n	$\frac{n\sigma_0^2\bar{x} + \sigma^2\mu_0}{\sigma^2 + n\sigma_0^2}, \frac{\sigma^2\sigma_0^2}{n\sigma_0^2 + \sigma^2}$
Normal (known μ)	Inverse Gamma	α, β	σ^2	$\sum(x_i - \mu)^2, n$	$\alpha + n/2, \beta + \frac{1}{2} \sum(x_i - \mu)^2$
Binomial (m trials)	Beta	α, β	p	s, n	$\alpha + s, \beta + nm - s$
Poisson	Gamma	α, β	λ	s, n	$\alpha + s, \beta + n$
Exponential	Gamma	α, β	λ	s, n	$\alpha + n, \beta + s$

Likelihood	Prior	Prior Parameters	Model Parameters	Sufficient Statistics	Posterior Parameters
Multinomial	Dirichlet	$\alpha \in \mathbb{R}^k$	$p \in \mathbb{R}^k$	s_j for each category j	$\alpha + s$
Normal	Normal-inverse gamma	$\mu_0, \nu, \alpha, \beta$	μ, σ	$\bar{x}, \sum(x_i - \bar{x})^2, n$	$\bar{x} + \frac{\nu\mu_0 + n\bar{x}}{\nu+n}, \nu+n, \alpha+\frac{n}{2}, \beta + \frac{1}{2} \sum_{i=1}^n (x_i - \bar{x})^2 + \frac{n\nu}{\nu+n} \frac{(\bar{x}-\mu_0)^2}{2}$

These conjugate relationships simplify Bayesian calculations by ensuring that the posterior distributions are in the same family as the priors.

3.11 Bayesian Computation: when conjugacy breaks

Conjugate priors are powerful because they keep the posterior in a familiar family and allow analytic updating. But most realistic Bayesian models are not conjugate: likelihoods can be non-Gaussian, priors can encode structure (sparsity, hierarchy), and predictors can enter through nonlinear functions. In those settings, the posterior still exists, but we cannot usually write it down in closed form.

The default remedy is approximation by simulation. The unifying idea is simple: if we can draw samples $\theta^{(1)}, \dots, \theta^{(N)} \sim p(\theta | y)$, then posterior summaries become Monte Carlo averages. For example, for any function $g(\theta)$,

$$\text{E}(g(\theta) | y) \approx \frac{1}{N} \sum_{n=1}^N g(\theta^{(n)}).$$

This is the same law-of-large-numbers logic discussed in Chapter 1, but applied to posterior distributions rather than to a known data-generating distribution.

3.11.1 Posterior Consistency and the Law of Large Numbers

The law of large numbers provides theoretical justification for Bayesian learning. As we collect more data, the posterior distribution concentrates around the true parameter value, regardless of the prior. This phenomenon, known as

posterior consistency, follows from the fact that the likelihood function—being a product of many terms—is dominated by the data for large samples.

Consider estimating the mean μ of a normal distribution from i.i.d. observations $x_1, \dots, x_n \sim N(\mu, \sigma^2)$ with a prior $\mu \sim N(\mu_0, \tau_0^2)$. The posterior mean is:

$$E(\mu | x_1, \dots, x_n) = \frac{\tau_0^{-2}\mu_0 + n\sigma^{-2}\bar{x}_n}{\tau_0^{-2} + n\sigma^{-2}}$$

As $n \rightarrow \infty$, the data term $n\sigma^{-2}\bar{x}_n$ dominates, and by the law of large numbers, $\bar{x}_n \rightarrow \mu$ almost surely. Thus:

$$E(\mu | x_1, \dots, x_n) \rightarrow \mu \text{ almost surely.}$$

The posterior concentrates at μ , regardless of the prior μ_0 . The prior matters for small samples but becomes negligible for large samples—a reassuring property that ensures different researchers with different priors eventually reach consensus as evidence accumulates.

3.11.2 Monte Carlo Methods

Monte Carlo simulation is also justified by the law of large numbers. To approximate an expectation $E(f(x))$ where $x \sim p(x)$, we draw i.i.d. samples $x_1, \dots, x_n \sim p(x)$ and compute:

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^n f(x_i) \approx E(f(x))$$

By the strong law of large numbers, $\hat{\mu}_n \rightarrow E(f(X))$ almost surely as $n \rightarrow \infty$, provided $E(|f(X)|) < \infty$. The approximation error decreases at rate $O_p(n^{-1/2})$ by the central limit theorem, giving us both convergence guarantees and quantifiable uncertainty.

The formal development of Monte Carlo methods emerged from the Manhattan Project during World War II, when Stanislaw Ulam, recovering from illness in 1946, realized that complex probability problems could be solved by simulating random processes rather than through analytical calculations. Ulam shared this insight with John von Neumann, who recognized its potential for solving neutron diffusion problems critical to nuclear weapons design and implemented the algorithms on early electronic computers like ENIAC. Nicholas Metropolis coined the term “Monte Carlo” as a reference to the Monaco casino, and the first unclassified paper appeared in 1949 (Metropolis and Ulam 1949). The 1953 Metropolis algorithm (Metropolis et al. 1953) extended the

method beyond simple averaging to sampling from complex distributions—precisely the situation in Bayesian inference—laying the groundwork for modern MCMC. The law of large numbers had existed for centuries, but only the combination of electronic computing and wartime urgency transformed this theoretical principle into the practical computational tool that underpins contemporary Bayesian statistics (Metropolis 1987).

3.11.3 Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) methods extend Monte Carlo simulation to settings where samples are dependent but ergodic. For an ergodic Markov chain with stationary distribution $\pi(x)$, the ergodic theorem guarantees:

$$\frac{1}{n} \sum_{i=1}^n f(x_i) \rightarrow E_\pi(f(x)) \quad \text{almost surely}$$

where x_i are states visited by the chain. This justifies using MCMC to approximate posterior expectations in Bayesian inference, even though consecutive samples are correlated, see Nicholas Polson (1996) for a formal analysis.

MCMC methods provide a powerful framework for sampling from complex distributions that cannot be sampled directly. The key insight is to construct a Markov chain $P(x, y)$ describing the transition probability from state x to state y , such that the desired posterior distribution π is the equilibrium (stationary) distribution of the chain.

Formally, a distribution π is stationary for a Markov chain with transition kernel $P(x, y)$ if:

$$\pi(y) = \int \pi(x)P(x, y)dx$$

Once we establish that π is stationary, we can generate samples $x^{(1)}, x^{(2)}, \dots$ by simulating the Markov chain for sufficiently many iterations. By the ergodic theorem, time averages along the chain converge to expectations under π :

$$\frac{1}{n} \sum_{i=1}^n f(x^{(i)}) \rightarrow E_\pi(f(x)) \quad \text{almost surely}$$

The remarkable feature of MCMC is that we only need to know the *ratio* of densities $\pi(y)/\pi(x)$, making it perfect for Bayesian inference where normalizing constants are often intractable. Two fundamental MCMC algorithms illustrate this principle:

Metropolis-Hastings Algorithm

The *Metropolis-Hastings* algorithm constructs a reversible Markov chain by proposing a candidate point y and accepting it with probability:

$$P(x, y) = \min\left(\frac{\pi(y)}{\pi(x)}, 1\right)$$

It is straightforward to verify that this chain is time-reversible (satisfies detailed balance):

$$\pi(x)P(x, y) = \min(\pi(y), \pi(x)) = \pi(y)P(y, x)$$

This implies that π satisfies the stationarity condition:

$$\sum_x \pi(x)P(x, y) = \pi(y)$$

and is therefore the equilibrium distribution of the chain.

Gibbs Sampler

The *Gibbs sampler* is a special case of the Metropolis-Hastings algorithm where the acceptance probability is always one. It is particularly useful for multivariate distributions where we can easily sample from conditional distributions. The Gibbs sampler cycles through coordinates, sampling each variable conditional on the current values of all others.

For a state vector $x = (x_1, \dots, x_p)$, one iteration of the Gibbs sampler updates:

$$x_j^{(t+1)} \sim \pi(x_j | x_1^{(t+1)}, \dots, x_{j-1}^{(t+1)}, x_{j+1}^{(t)}, \dots, x_p^{(t)})$$

The Gibbs sampler is time-reversible due to the Clifford-Hammersley theorem, which guarantees that the joint distribution is the unique stationary distribution. This makes the Gibbs sampler a special case of the multivariate coordinate-by-coordinate sampler where the acceptance probability is always one.

Both algorithms transform the problem of computing complex integrals into a problem of simulating a carefully constructed stochastic process. The variance-mean mixture representations discussed in Chapter 17 leverage exactly this principle: by introducing auxiliary variables, they convert intractable posterior distributions into forms amenable to Gibbs sampling with conjugate conditional distributions.

Posterior predictive simulation is the recurring workflow that connects computation back to modeling. To forecast or to check fit, we first draw parameters from the posterior and then simulate replicated or future data:

$$\theta^{(n)} \sim p(\theta | y), \quad \tilde{y}^{(n)} \sim p(\tilde{y} | \theta^{(n)}).$$

Repeating this produces an empirical approximation to the posterior predictive distribution $p(\tilde{y} | y)$ and makes uncertainty propagation concrete: parameter uncertainty becomes predictive uncertainty.

Later chapters return to this pattern in different guises: sampling-based Bayesian inference (e.g., for logistic regression), approximate inference for scalable models, and simulation-based decision making. The key takeaway here is that conjugacy is a convenience, not a requirement; when the algebra ends, simulation provides the continuation.



4

Utility, Risk and Decisions

“I would never die for my beliefs because I might be wrong.” – Bertrand Russell

Statistical decision theory asks: given what the data tells us, what should we do? The problem typically splits in two. First, learn something from data—estimate parameters, fit a model. Second, use that learning to choose an action. In finance, this means estimating means and covariances from historical data, then constructing a portfolio. In statistics, it means evaluating which estimator or hypothesis test performs best under a given criterion.

The examples in the previous chapter focused on learning about unknown quantities from data, expressed through posterior distributions. In many applications, however, we must turn a posterior into an action: approve a loan, deploy a model, choose a medical treatment, or ship a product variant. This step requires an additional ingredient beyond probability modeling: a loss function that formalizes what we mean by a good or bad decision.

For a parameter θ and a decision (or estimate) a , a loss $\mathcal{L}(\theta, a)$ assigns a numerical penalty to choosing a when the truth is θ . Given data y , Bayesian decision making compares actions using posterior expected loss,

$$\text{E}(\mathcal{L}(\theta, a) \mid y).$$

The optimal Bayesian action is the posterior Bayes rule

$$a^*(y) = \arg \min_a \text{E}(\mathcal{L}(\theta, a) \mid y).$$

Several estimators that are common in statistics can be interpreted as Bayes rules under simple choices of loss. For example, the posterior mean minimizes squared error loss $\mathcal{L}(\theta, a) = (\theta - a)^2$, while the posterior median minimizes absolute error loss $\mathcal{L}(\theta, a) = |\theta - a|$. Under 0–1 loss for point classification, the posterior mode becomes optimal. This perspective makes point estimation a special case of a more general principle: choose the action that is optimal under the posterior distribution and the costs of being wrong.

This chapter develops this decision-theoretic framework systematically, connecting loss functions to utility, risk, and optimal actions under uncertainty.

4.1 Expected Utility

Let P, Q be two possible *risky gambles* or probability bets. An agent's preferences can then be specified as an ordering on probability bets where we write P is preferred to Q as $P \succeq Q$ and indifference as $P \sim Q$. A compound or mixture bet is defined by the probability assignment $pP + (1 - p)Q$ for a prospect weight $0 \leq p \leq 1$.

Ramsey-de Finetti-Savage show that if an agent's preferences satisfy a number of plausible axioms – completeness, transitivity, continuity and independence – then they can be represented by the expectation of a utility function. The theory is a *normative* one and not necessarily *descriptive*. It suggests how a rational agent should formulate beliefs and preferences and not how they actually behave.

This representation of preferences in terms of expected utility $U(P)$ of a risky gamble is then equivalent to

$$P \succeq Q \iff U(P) \geq U(Q)$$

Therefore, the higher the value taken by the utility function the more the gamble is preferred. Specifically, the axioms lead to existence of expected utility and uniqueness of probability.

The two key facts then are uniqueness of probability and existence of expected utility. Formally,

1. If $P \succeq R \succeq Q$ and $wP + (1 - w)Q \sim R$ then w is unique.
2. There exists an expected utility $U(\cdot)$ such that $P \succeq Q \iff U(P) \geq U(Q)$. Furthermore

$$U(wP + (1 - w)Q) = wU(P) + (1 - w)U(Q)$$

for any P, Q and $0 \leq w \leq 1$.

This implies that U is additive and it is also unique up to affine transformation.

Proof: If w is not unique then $\exists w_1$ such that $w_1 P + (1 - w_1)Q \sim R$. Without loss of generality assume that $w_1 < w$ and so $0 < w - w_1 < 1 - w_1$. However, we can write the bet Q as

$$Q = \left(\frac{w - w_1}{1 - w_1} \right) Q + \left(\frac{1 - w}{1 - w_1} \right) Q$$

By transitivity, as $P \succeq Q$ we have

$$\left(\frac{w - w_1}{1 - w_1} \right) P + \left(\frac{1 - w}{1 - w_1} \right) Q \succeq Q$$

However,

$$wP + (1 - w)Q = w_1 P + (1 - w_1) \left(\left(\frac{w - w_1}{1 - w_1} \right) P + \left(\frac{1 - w}{1 - w_1} \right) Q \right)$$

implying by transitivity that

$$wP + (1 - w)Q \succeq w_1 P + (1 - w_1)Q$$

which is a contradiction.

This can be used together with the axioms to then prove the existence and uniqueness of a utility function.

Theorem 4.1 (Uniqueness of Utility). *If V is any other function satisfying these results then V is an affine function of U .*

Proof. If $\forall P, Q$ we have $P \sim Q$, then define $u(P) \equiv 0$. Hence suppose that there exists $S \succ T$. Define $U(S) = 1$ and $U(T) = 0$. For any $P \in \mathcal{P}$ there are five possibilities: $P \succ T$ or $P \sim S$ or $S \succ P \succ T$ or $P \sim T$ or $T \succ P$.

In the first case define $1/U(P)$ to be the unique p (see previous theorem) defined by $pP + (1 - p)T \sim S$. In the second case, define $U(P) = 1$. In the third, there exists a unique q with $qS + (1 - q)T \sim P$ and then define $U(P) = q$. In the fourth case, define $U(P) = 0$ and finally when $T \succ P$ there exists a unique r with $rS + (1 - r)P \sim T$ and then we define $U(P) = -r/(1 - r)$.

Then check that $U(P)$ satisfies the conditions. See Savage (1954), Ramsey (1927) and de Finetti (1931) \square

Other interesting extensions: how do people come to a consensus (DeGroot 1974; Morris 1994, 1996). Ramsey (1926) observation that if someone is willing to offer you a bet then that's conditioning information for you. All probabilities are conditional probabilities.

If the bet outcome x is a monetary value, then the utility functions $x, x^2, \sqrt{x}, \ln x$ are all monotonically increasing (the more the better). However, the utility function x^2 is convex and the utility function $\ln x$ is concave. The concavity of the utility function implies that the agent is risk averse and the convexity implies that the agent is risk seeking.

Example 4.1 (Saint Petersburg Paradox). The Saint Petersburg paradox is a concept in probability and decision theory that was first introduced by Daniel Bernoulli in 1738. It revolves around the idea of how individuals value risky propositions and how those valuations may not align with classical expected utility theory.

The paradox is named after the city of Saint Petersburg, where the problem was formulated. Here's a simplified version of the paradox:

Imagine a gambling game where a fair coin is flipped repeatedly until it lands on heads. The payoff for the game is 2^N , where N is the number of tosses needed for the coin to land on heads. The expected value of this game, calculated by multiplying each possible payoff by its probability and summing the results, is infinite:

$$E(X) = \frac{1}{2} \cdot 2 + \frac{1}{4} \cdot 4 + \frac{1}{8} \cdot 8 + \dots = \infty$$

This means that, in theory, a rational person should be willing to pay any finite amount to play this game, as the expected value is infinite. However, in reality, most people would be unwilling to pay a large amount to play such a game.

The paradox arises because traditional expected utility theory assumes that individuals make decisions based on maximizing their expected gain. Bernoulli argued that people do not maximize expected monetary value but rather expected utility, where utility is a subjective measure of satisfaction or happiness. He proposed that individuals exhibit diminishing marginal utility for wealth, meaning that the additional satisfaction gained from an extra unit of wealth decreases as total wealth increases.

In the case of the Saint Petersburg paradox, although the expected monetary value is infinite, the utility gained from each additional dollar diminishes rapidly, leading to a reluctance to pay large amounts to play the game.

In modern decision theory and economics, concepts like diminishing marginal utility and expected utility are fundamental in understanding how individuals make choices under uncertainty and risk. The Saint Petersburg paradox highlights the limitations of relying solely on expected monetary value in explaining human behavior in such situations.

One common approach is to consider aspects of potential players, such as their possible risk aversion, available funds, etc., through a utility function $U(x)$. Applying a utility function in this situation means changing our focus to the quantity

$$E[U(X)] = \sum_{k=1}^{\infty} 2^{-k} U(2^k).$$

Some examples of utility functions are,

- $U(x) = W_0(1 - x^{-\alpha})$, $\alpha > 0$, which gives an expected utility of $W_0 \left(1 - \frac{1}{2^{\alpha+1}-1}\right)$
- Log utility, $U(x) = \log(x)$, with expected value $2 \log(2)$.

Notice that after obtaining an expected utility value, you'll have to find the corresponding reward/dollar amount.

For the log utility case, we need to find the certain dollar amount x^* that provides the same utility as playing the game. Setting $U(x^*) = 2 \log(2)$, we solve:

$$\log(x^*) = 2 \log(2) = \log(2^2) = \log(4)$$

which gives $x^* = 4$. Therefore, under log utility, a rational player would be willing to pay at most \$4 to play the Saint Petersburg game, despite its infinite expected monetary value. This is a dramatic reduction from infinity and demonstrates how risk aversion (captured by the concave log utility function) resolves the paradox.

Similarly, for the power utility $U(x) = W_0(1 - x^{-\alpha})$ with $\alpha > 0$, we have an expected utility of $W_0(1 - \frac{1}{2^{\alpha+1}-1})$. To find the certainty equivalent x^* , we solve:

$$W_0(1 - (x^*)^{-\alpha}) = W_0\left(1 - \frac{1}{2^{\alpha+1} - 1}\right)$$

which simplifies to $(x^*)^{-\alpha} = \frac{1}{2^{\alpha+1}-1}$, giving:

$$x^* = (2^{\alpha+1} - 1)^{1/\alpha}$$

For example, with $\alpha = 1$, we get $x^* = (2^2 - 1)^1 = 3$ dollars. As risk aversion increases (larger α), the certainty equivalent further decreases.

Bernoulli's resolution of this paradox was a watershed moment: it shifted the focus from objective *expected value* (which is infinite here) to subjective *expected utility*. This idea—that people value money non-linearly—became the cornerstone of the modern economic theory of risk.

Example 4.2 (Newcomb's Problem). Newcomb's problem, introduced by physicist William Newcomb and first analyzed philosophically by Robert Nozick in 1969, remains one of the most debated puzzles in decision theory. It exposes a fundamental tension between two seemingly reasonable principles for making decisions under uncertainty.

Following Seidenfeld (1984), the canonical formulation involves a choice between two options: *option*—to take the contents of an opaque box (one-box); and *option*—to take the contents of the opaque box plus the \$1,000 in the transparent box (two-boxes). The opaque box contains either \$1,000,000 or nothing, depending upon a demon's earlier prediction that the choice is for one-box or for two-boxes. The decision matrix summarizes the choice problem:

	Demon predicts correctly	Demon predicts incorrectly
Option (one-box)	\$1,000,000	\$0
Option (two-boxes)	\$1,000	\$1,001,000

This Savage-styled formulation frames the demon's accuracy as the state of the world. Seidenfeld notes the convenience: the verity of the demon's prediction affords a binary partition of states that can be treated as probabilistically act-independent. What is striking is that neither option dominates the other—option wins when the demon is correct, option 2 wins when incorrect. Yet taking two boxes always yields \$1,000 more *for any fixed content of the opaque box*. We assume the agent takes no interest in the consequences apart from monetary outcomes (no intrinsic desire to frustrate the demon). The twist is to add a constraint that the chooser admits the demon is a very accurate predictor, with conditional probability given the agent's choice:

$$P(\text{demon predicts option}_i \mid \text{option}_i) = 1 - \varepsilon \approx 1.$$

Then the conditional expected utility of option exceeds that of option — against the recommendation afforded by appeal to dominance. As Seidenfeld notes, this exposes a “deep rooted problem in normative expected utility theory” arising from “the difference between probabilistic dependence and causal dependence across act-state pairs.”

Two principles from decision theory give conflicting advice:

Conditional Expected Utility: Given the predictor's high accuracy, the expected value of one-boxing is approximately $(1 - \varepsilon) \times 1,000,000 + \varepsilon \times 0 \approx \$1,000,000$, while two-boxing yields approximately $\varepsilon \times 1,001,000 + (1 - \varepsilon) \times 1,000 \approx \$1,000$. One-boxing maximizes expected utility.

Strategic Dominance: The predictor has already made the prediction and placed the money. Whatever is in the opaque box is “fixed and determined” when the agent chooses. Attention to the causal efficacy of the choice makes the dominating option right.

The limiting case illuminates the paradox. Consider an *infallible* demon where $\varepsilon = 0$. Then by standard decision theory, the problem becomes one of choice under certainty—there is no risk involved. Taking option and getting \$1,000,000, or taking option and getting \$1,000, are the only possibilities. Do causal decision theorists advocate option when it is certain to yield \$1,000 whereas option is certain to yield \$1,000,000? As Seidenfeld argues, “To recommend two-boxes in this case is to contravene more than subjective expected utility theory—it is to overturn what is non-controversial by all standard accounts of choice, from expected utility to minimax theory to the very principle of dominance.”

Seidenfeld also raises a compelling third-person version: if your friend faces the Newcomb choice, what are your betting odds that conditional on option your friend becomes a millionaire? By the weakest principles of conditionalization, these are fixed by the problem's constraints. What is your fair-market price for your friend's choice? Given your friend chooses one-box, is there any reason to pass up the bargain to buy the proceeds for \$10,000?

The problem touches on deep questions about causation and free will. Causal decision theory holds that we should choose actions based on their causal consequences—and since our choice *cannot causally affect* what is already in the opaque box, we should two-box. Evidential decision theory counters that we should choose actions that provide the best *evidence* about outcomes—and one-boxing is strong evidence that the box contains the million.

Newcomb's problem has practical relevance beyond philosophy. In game theory, similar structures arise when facing opponents who can anticipate your strategy. In artificial intelligence, it relates to how agents should reason when their decision procedures are transparent to other agents. The problem also connects to debates about deterrence in international relations, where the credibility of threats depends on whether adversaries believe you would follow through—a prediction about your future choice that influences present outcomes.

A 2020 survey of professional philosophers found a modest plurality (39% vs. 31%) favoring two-boxing, though the debate remains very much alive. The problem continues to generate new insights into the foundations of rational decision-making and the relationship between choice and causation.

Now, consider a more general situation, when you have two gambles 1: get P_1 for sure, 2: get $P_2 = P_1 + k$ and $P_3 = P_1 - k$ with probability 1/2. Then we will compare the utility of those gambles

$$\frac{1}{2}U(P_2) + \frac{1}{2}U(P_3) \text{ and } U(P_1).$$

If the utility function is linear then we should be indifferent between the two gambles. However, if the utility function is concave then we should prefer the sure thing. This is known as the *certainty effect*.

$$\frac{1}{2}U(P_2) + \frac{1}{2}U(P_3) < U(P_1).$$

The usual situation can be described as follows. Let Ω be a finite set of possible outcomes with $\Omega = \{\omega_1, \dots, \omega_n\}$. Let P_i be the consequence that assigns one to outcome ω_i and zero otherwise and let $P = (p_1, \dots, p_n)$ assign probability p_i to outcome ω_i . Then we can write the expected utility, $U(P)$, of the gamble P as

$$U(P) = \sum_{i=1}^n p_i U(P_i).$$

That is, the utility of P is the expected value of a random variable W (wealth) that takes the value $U(P_i)$ if the outcome is ω_i . Therefore, we can write $U(P) = \mathbb{E}_P(U(W))$.

This leads us to the notion of *risk aversion* and a categorization of agents according to their risk tolerance: the agent is said to be

1. *Risk Averse* if $\mathbb{E}_P(U(W)) \leq U(\mathbb{E}_P(W))$
2. *Risk Neutral* if $\mathbb{E}_P(U(W)) = U(\mathbb{E}_P(W))$
3. *Risk Seeking* if $\mathbb{E}_P(U(W)) \geq U(\mathbb{E}_P(W))$

Here we assume that these hold for all probabilities and random variables. Risk aversion is equivalent to the agent having concave utility and risk seeking convex.

Example 4.3 (Risk Aversion). Consider the family of Constant Relative Risk Aversion (CRRA) utility functions. This family includes the log-utility (Kelly)

$$U(W) = \log(W),$$

and the power utility

$$U(W) = \frac{W^{1-\gamma} - 1}{1 - \gamma}.$$

The log-utility is a limiting case of the power utility when $\gamma \rightarrow 1$. The parameter γ controls the curvature of the function and is known as the coefficient of relative risk aversion:

$$R(W) = -W \frac{U''(W)}{U'(W)} = \gamma.$$

Higher values of γ imply greater concavity, meaning the agent is more risk-averse and requires a higher risk premium to accept a gamble.

- $\gamma \rightarrow 0$: Risk neutral (linear utility).
- $\gamma = 0.5$: Moderate risk aversion (less than log).
- $\gamma = 1$: Log utility (Kelly criterion).
- $\gamma > 1$: High risk aversion (e.g., fractional Kelly).

Figure 4.1 demonstrates the effect of γ . As γ increases, the utility function becomes more curved (“bends” more). This means that losses (moving left from $W = 1$) hurt much more in utility terms than equivalent gains (moving right) help.

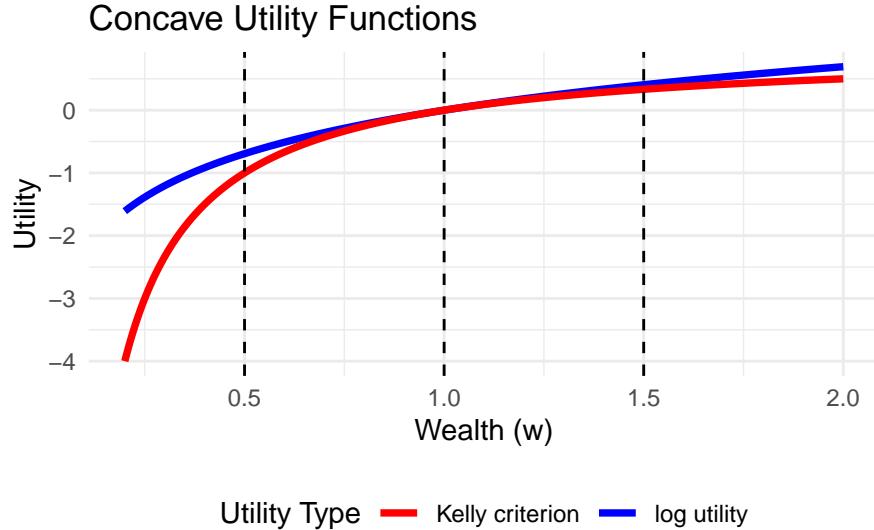


Figure 4.1: Utility functions for different risk aversion parameters.

To see this concretely, consider a simple gamble with two equally likely outcomes: wealth $w_1 = 0.5$ and $w_2 = 1.5$. The expected wealth is $E[W] = 0.5(0.5) + 0.5(1.5) = 1.0$.

For a risk-neutral agent ($\gamma = 0$), the utility is linear, so the value of the gamble is exactly the expected wealth (1.0). However, for a risk-averse agent ($\gamma > 0$), the utility of the expected wealth is greater than the expected utility of the gamble. The **Certainty Equivalent (CE)** is the guaranteed amount of wealth that provides the same utility as the risky gamble. The difference between the expected wealth and the certainty equivalent is the **Risk Premium**—the amount the agent is willing to “pay” (forgo in expected value) to avoid the risk.

The table below shows how the Certainty Equivalent decreases and the Risk Premium increases as the risk aversion parameter γ grows.

Table 4.2: Impact of risk aversion on the valuation of a gamble with outcomes {0.5, 1.5}.

Risk Aversion (γ)	Certainty Equivalent	Risk Premium
0.0	1.00	0.000
0.5	0.93	0.067
1.0	0.87	0.134
2.0	0.75	0.250

Risk Aversion (γ)	Certainty Equivalent	Risk Premium
4.0	0.62	0.378

Example 4.4 (Kelly Criterion). The Kelly criterion has been used effectively by many practitioners. Ed Thorp, in his book *Beat the Dealer*, pioneered its use in blackjack and later applied it to investing in financial markets. Since then, many market participants, such as Jim Simons, have stressed the importance of this money management approach. The criterion's application extends to other domains: Phil Laak described its use for bet sizing in a game-theoretic approach to poker, and Bill Benter applied it to horse racing. Stewart Ethier provided a mathematical framework for multiple outcomes and analyzed a “play the winner” rule in roulette. Claude Shannon also developed a system to detect and exploit unintentionally biased roulette wheels, an endeavor chronicled in the book *The Eudaemonic Pie*.

Suppose you have \$1000 to invest. With probability 0.55 you will win whatever you wager and with probability 0.45 you lose whatever you wager. What's the proportion of capital that leads to the fastest compounded growth rate?

Quoting Kelly (1956), the exponential rate of growth, G , of a gambler's capital is

$$G = \lim_{T \rightarrow \infty} \frac{1}{T} \log_2 \frac{W_T}{W_0}$$

for initial capital W_0 and capital after T bets W_T .

Under the assumption that a gambler bets a fraction of his capital, ω , each time, we use

$$W_T = (1 + \omega)^W (1 - \omega)^L W_0$$

where W and L are the number of wins and losses in N bets. We get

$$G = p \log_2 (1 + \omega) + q \log_2 (1 - \omega)$$

in which the limit(s) of $\frac{W}{N}$ and $\frac{L}{N}$ are the probabilities p and q , respectively.

This also comes about by considering the sequence of i.i.d. bets with

$$p(X_t = 1) = p \quad \text{and} \quad p(X_t = -1) = q = 1 - p$$

We want to find an optimal allocation ω^* that maximizes the expected long-run growth rate:

$$\begin{aligned} \max_{\omega} \mathbb{E}(\ln(1 + \omega W_T)) &= p \ln(1 + \omega) + (1 - p) \ln(1 - \omega) \\ &\leq p \ln p + q \ln q + \ln 2 \quad \text{and} \quad \omega^* = p - q \end{aligned}$$

The solution is $w^* = 0.55 - 0.45 = 0.1$.

Both approaches give the same optimization problem, which, when solved, give the optimal fraction rate $\omega^* = p - q$, thus, with $p = 0.55$, the optimal allocation is 10% of capital.

We can generalize the rule to the case of asymmetric payouts (a, b) . Then the expected utility function is

$$p \ln(1 + b\omega) + (1 - p) \ln(1 - a\omega)$$

The optimal solution is

$$\omega^* = \frac{bp - aq}{ab}$$

If $a = b = 1$ this reduces to the pure Kelly criterion.

A common case occurs when $a = 1$ and market odds $b = O$. The rule becomes

$$\omega^* = \frac{p \cdot O - q}{O}.$$

Let's consider another scenario. You have two possible market opportunities: one where it offers you 4/1 when you have personal odds of 3/1 and a second one when it offers you 12/1 while you think the odds are 9/1.

In expected return these two scenarios are identical both offering a 33% gain. In terms of maximizing long-run growth, however, they are not identical.

Table 4.3 shows the Kelly criterion advises an allocation that is twice as much capital to the lower odds proposition: 1/16 weight versus 1/40.

Table 4.3: Kelly rule

Market	You	Δ	ω^*
4/1	3/1	1/4	1/16
12/1	9/1	1/10	1/40

The optimal allocation $\omega^* = (pO - q)/O$ is

$$\frac{(1/4) \times 4 - (3/4)}{4} = \frac{1}{16} \quad \text{and} \quad \frac{(1/10) \times 12 - (9/10)}{12} = \frac{1}{40}.$$

Note, that although the expected return is the same, the risk is different. The first gamble has a higher variance than the second gamble.

Power utility and log-utilities allow us to model constant relative risk aversion (CRRA). The main advantage is that the optimal rule is unaffected by wealth effects. The CRRA utility of wealth takes the form

$$U_\gamma(W) = \frac{W^{1-\gamma} - 1}{1 - \gamma}$$

The special case $U(W) = \log(W)$ for $\gamma = 1$. This leads to a myopic Kelly criterion rule.

4.2 Historical Perspectives on Optimal Decision Making

The principles of optimal decision-making under uncertainty have deep historical roots that extend well beyond modern portfolio theory. Understanding this intellectual lineage enriches our appreciation of the Kelly criterion and provides important lessons about risk management.

4.2.1 De Finetti's Insurance Problem

In 1940, more than a decade before the canonical papers on portfolio theory, the Italian mathematician Bruno de Finetti addressed the problem of optimal retention in insurance (de Finetti 1940). An insurance company holding a portfolio of policies faces a crucial decision: how much of each policy's risk should it retain versus reinsuring? This is essentially an asset allocation problem where the “assets” are insurance policies with uncertain payouts.

De Finetti formulated this as a mean-variance optimization problem with constraints $0 \leq f_i \leq 1$ on the retention fractions, where f_i represents the fraction of policy i kept for the company's own account. For uncorrelated risks, he proved that the optimal retention for each policy should be

$$f_i = A \frac{k_i}{\sigma_i^2}$$

where k_i is the margin of gain from the policy, σ_i^2 is its variance, and A is a constant determined by the company's risk tolerance. This elegant result states that retention should be proportional to profitability and inversely proportional to variance—precisely the Kelly criterion's risk-return trade-off.

Remarkably, de Finetti extended this analysis to the case of *correlated* risks, recognizing that insurance claims (like stock returns) are often positively correlated due to common factors such as economic conditions or natural disasters. In this case, the problem becomes significantly more complex. De Finetti showed that with correlations, one must solve a system of linear equations to determine optimal retentions, and he outlined—though did not fully solve—the general algorithm for finding the efficient frontier when risks are correlated.

De Finetti also introduced what we now call the “index of stability” for the long-run solvency of an insurance company. If G is the guarantee fund, m the

expected annual gain, and σ^2 the variance of gains, then the probability of ruin P over the long run is approximately

$$P \approx e^{-\xi} \quad \text{where} \quad \xi = \frac{2mG}{\sigma^2}$$

This shows that stability increases linearly with the profit margin and capital reserves, but decreases with the square of risk—a fundamental relationship that applies equally to trading and investment.

4.2.2 Markowitz's Recognition

It was not until 2006 that Harry Markowitz, in a paper titled “De Finetti Scoops Markowitz” (Markowitz 2006), brought widespread attention to de Finetti’s pioneering work. Markowitz acknowledged that de Finetti had essentially proposed mean-variance analysis with correlated risks in 1940, more than a decade before the famous 1952 papers by Markowitz himself and A.D. Roy.

Despite their mathematical similarity, the two formulations differ in important ways that reflect their distinct applications. Markowitz’s portfolio problem includes a budget constraint $\sum X_i = 1$ (fractions invested must sum to one) and seeks either to maximize expected return for a given level of variance, or equivalently to minimize variance for a given expected return. In contrast, de Finetti’s reinsurance problem has no budget constraint—the company can choose any retention fractions $0 \leq f_i \leq 1$ independently for each policy. His objective is to minimize risk (variance) subject to maintaining a fixed level of expected profit from the portfolio. Both formulations trace out an efficient frontier, but they arrive there from different starting points: Markowitz optimizes the trade-off between return and risk along a budget line, while de Finetti optimizes retention to minimize risk for a desired profit margin. Mathematically, both can be solved by minimizing $L = \frac{1}{2}V_P - wE_P$ for various weights w , but the constraint sets differ fundamentally.

Markowitz noted that while de Finetti solved the reinsurance problem completely for uncorrelated risks and outlined key properties for the correlated case, he did not provide a complete computational algorithm for the latter. Markowitz showed that the de Finetti problem is a special case of problems solvable by the Critical Line Algorithm (CLA) he developed in 1956. The CLA efficiently traces out the entire efficient frontier by moving along connected line segments in (X, η, λ) space, where $\eta_i = \partial L / \partial X_i$ measures the marginal change in the objective function.

An interesting feature of the de Finetti problem, which Markowitz corrected, concerns the final segment of the efficient frontier. De Finetti conjectured that the path from maximum expected return to minimum variance always approaches the zero portfolio from the interior of the feasible region. Markowitz

proved this is not always true—with correlated risks, some policies may already be at zero retention on the final segment. However, Markowitz established a “correct last segment theorem”: once no policy is fully retained (all $X_i < 1$), that segment must lead directly to the minimum variance portfolio.

4.2.3 Dynamic Kelly and Bellman’s Principle

While the static Kelly criterion provides optimal allocation for a single period, real investors face sequential decisions where today’s choices affect tomorrow’s opportunities. This is where dynamic programming enters the picture. Richard Bellman’s Principle of Optimality (1957) states that “whatever the initial state and initial decision, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.”

For asset allocation, this leads to a fundamental recursion. If $Q_t(s, a)$ denotes the value of taking action a in state s at time t , then

$$Q_t(s, a) = \mathbb{E}[r(a, s, S_{t+1}) + V(S_{t+1}) \mid s, a]$$

where $V(s) = \max_a Q_t(s, a)$ is the value function. In the i.i.d. case with logarithmic utility, the myopic Kelly rule remains optimal—future learning opportunities don’t change today’s allocation. This remarkable simplification explains the Kelly criterion’s enduring popularity.

However, when the investor can *learn* about the underlying probabilities, the dynamic problem becomes more sophisticated (Jacquier and Polson 2013). Consider a Bernoulli-Beta setting where success probability p is unknown and the investor starts with prior $p \sim \text{Beta}(\alpha, \beta)$. The myopic Kelly rule would invest nothing if $\alpha < \beta$ (unfavorable prior odds). Yet the dynamic Bellman solution shows that even a risk-averse investor with $\alpha < \beta$ should invest a small positive amount!

The reason is the *option value of learning*: by investing even when current odds seem unfavorable, the investor gains information that may reveal the true environment is actually favorable. For instance, with a uniform prior Beta(1, 1) and 10 remaining periods, an investor might optimally allocate 11.7% even after observing one failure (giving posterior Beta(1, 2) with expected odds 1:2 against). The potential to learn trumps the current negative expectation. This insight has profound implications for exploration-exploitation trade-offs in uncertain environments.

4.2.4 Livermore’s Trading Wisdom

The legendary trader Jesse Livermore, whose methods were chronicled in the 1923 classic *Reminiscences of a Stock Operator*, articulated trading principles

that align remarkably well with Bellman's optimal control framework (Nick Polson and Witte 2014). Two of his famous maxims are particularly revealing:

"Profits always take care of themselves, but losses never do." In the Bellman framework, suppose an investor holds a long position because $Q_t(s, a_L) > Q_t(s, a_N)$ (long is better than neutral). If the position subsequently loses money, the trader faces a choice. Under unchanged beliefs, the same logic would suggest staying long. But Livermore's insight is that the price movement itself contains information. By Bayes' theorem, observing a loss should update beliefs such that $Q_{t+1}^{q \cup \{x_t\}}(s', a_L) < Q_{t+1}^{q \cup \{x_t\}}(s', a_N)$, where $q \cup \{x_t\}$ represents updated beliefs incorporating the price change. The optimal action is to exit. Conversely, a winning position confirms the original thesis—"it takes care of itself."

"Never average down." The practice of doubling up on losing positions seems to violate basic decision theory. If averaging down is optimal, it means $Q_t(s', a_L) > Q_t(s', a_N)$ after a loss. But if we've updated beliefs based on the adverse price movement, this typically cannot hold unless we've received external information beyond what the market revealed. Livermore put it bluntly: "Why send good money after bad?"

These trading heuristics embody Bellman's principle that the only thing that matters for optimal decision-making is the current state and what's optimal going forward—not what you paid, what you hoped for, or what you "should" have made. The past is irrelevant except insofar as it informs beliefs about the future. This perspective removes emotional obstacles to sound decision-making: the sunk cost of past losses, the pride in being "right," or the fear of admitting error.

These four contributions—de Finetti's insurance mathematics, Markowitz's portfolio theory, Bellman's dynamic programming, and Livermore's practical wisdom—represent complementary perspectives on the same fundamental problem of optimal decision-making under uncertainty. De Finetti provided the mathematical foundation showing that mean-variance optimization with betting constraints yields the Kelly-type allocation rules. Markowitz extended and formalized these methods, showing how to efficiently compute solutions even with correlations. Bellman revealed when and why sequential decision problems allow or forbid simple myopic solutions. And Livermore demonstrated that these theoretical principles have practical and timeless value in actual markets.

The common thread: optimal allocation depends on the *ratio of expected excess return to variance*, must account for correlations, must adapt to new information, and should never let past outcomes cloud judgment about future prospects. Whether managing an insurance portfolio in 1940, building an investment strategy in 1952, or trading in markets today, these principles remain the bedrock of rational decision-making.

4.3 Statistical Decisions and Risk

While Bernoulli's work laid the foundation for expected utility, it was in the mid-20th century that **L.J. Savage** and **John von Neumann/Oskar Morgenstern** formalized these concepts into a rigorous mathematical decision theory. This framework provides the "rational" standard against which human behavior is often measured.

The statistical decision making problem can be posed as follows. A decision maker (you) has to choose from a set of decisions or acts. The consequences of these decisions depend on an unknown state of the world. Let $d \in \mathcal{D}$ denote the decision and $\theta \in \Theta$ the state of the world. As an example, think of θ as the unknown parameter and the decision as choosing a parameter estimation or hypothesis testing procedure. To provide information about the parameter, the decision maker obtains a sample $y \in \mathcal{Y}$ that is generated from the likelihood function $p(y|\theta)$. The resulting decision depends on the observed data, is denoted as $d(y)$, and is commonly called the decision rule.

To make the decision, the decision maker uses a "loss" function as a quantitative metric to assesses the consequences or performance of different decisions. For each state of the world θ , and decision d , $\mathcal{L}(\theta, d)$ quantifies the "loss" made by choosing d when the state of the world is θ . Common loss functions include a quadratic loss, $\mathcal{L}(\theta, d) = (\theta - d)^2$, an absolute loss, $\mathcal{L}(\theta, d) = |\theta - d|$, and a 0–1 loss,

$$\mathcal{L}(\theta, d) = L_0 1_{[\theta \in \Theta_0]} + L_1 1_{[\theta \in \Theta_1]}.$$

For Bayesians, the utility function provides a natural loss function. Historically, decision theory was developed by classical statisticians, thus the development in terms of "objective" loss functions instead of "subjective" utility.

Classical decision theory takes a frequentist approach, treating parameters as "fixed but unknown" and evaluating decisions based on their population properties. Intuitively, this thought experiment entails drawing a dataset y of given length and applying the same decision rule in a large number of repeated trials and averaging the resulting loss across those hypothetical samples. Formally, the classical risk function is defined as

$$R(\theta, d) = \int_{\mathcal{Y}} \mathcal{L}[\theta, d(y)] p(y|\theta) dy = \mathbb{E} [\mathcal{L}[\theta, d(y)] | \theta].$$

Since the risk function integrates over the data, it does not depend on a given observed sample and is therefore an ex-ante or a-priori metric. In the case of

quadratic loss, the risk function is the mean-squared error (MSE) and is

$$\begin{aligned} R(\theta, d) &= \int_y [\theta - d(y)]^2 p(y|\theta) dy \\ &= \mathbb{E} [(d(y) - E[d(y)|\theta])^2 | \theta] + \mathbb{E} [(E[d(y)|\theta] - \theta)^2 | \theta] \\ &= Var(d(y)|\theta) + [bias(d(y) - \theta)]^2 \end{aligned}$$

which can be interpreted as the bias of the decision/estimator plus the variance of the decision/estimator. Common frequentist estimators choose unbiased estimators so that the bias term is zero, which in most settings leads to unique estimators.¹

The goal of the decision maker is to minimize risk. Unfortunately, rarely is there a decision that minimizes risk uniformly for all parameter values. To see this, consider a simple example of $y \sim N(\theta, 1)$, a quadratic loss, and two decision rules, $d_1(y) = 0$ or $d_2(y) = y$. Then, $R(\theta, d_1) = \theta^2$ and $R(\theta, d_2) = 1$. If $|\theta| < 1$, then $R(\theta, d_1) < R(\theta, d_2)$, with the ordering reversed for $|\theta| > 1$. Thus, neither rule uniformly dominates the other.

One way to deal with the lack of uniform domination is to use the minimax principle: first maximize risk as function of θ ,

$$\theta^* = \arg \max_{\theta \in \Theta} R(\theta, d),$$

and then minimize the resulting risk by choosing a decision:

$$d_m^* = \arg \min_{d \in \mathcal{D}} [R(\theta^*, d)].$$

The resulting decision is known as a minimax decision rule. The motivation for minimax is game theory, with the idea that the statistician chooses the best decision rule against the other player, mother nature, who chooses the worst parameter.

The Bayesian approach treats parameters as random and specifies both a likelihood and prior distribution, denoted here by $\pi(\theta)$. The Bayesian decision maker recognizes that both the data and parameters are random, and accounts for both sources of uncertainty when calculating risk. The Bayes risk is defined as

$$\begin{aligned} r(\pi, d) &= \int_{\Theta} \int_y \mathcal{L}[\theta, d(y)] p(y|\theta) \pi(\theta) dy d\theta \\ &= \int_{\Theta} R(\theta, d) \pi(\theta) d\theta = \mathbb{E}_{\pi}[R(\theta, d)], \end{aligned}$$

¹In most settings, since the unbiased estimator is unique, the minimum variance unbiased estimator is the minimum of a set containing a single estimator.

and thus the Bayes risk is an average of the classical risk, with the expectation taken under the prior distribution. The Bayes decision rule minimizes expected risk:

$$d_{\pi}^* = \arg \min_{d \in \mathcal{D}} r(\pi, d).$$

The classical risk of a Bayes decision rule is defined as $R(\theta, d_{\pi}^*)$, where d_{π}^* does not depend on θ or y . Minimizing expected risk is consistent with maximizing posterior expected utility or, in this case, minimizing expected loss. Expected posterior risk is

$$r(\pi, d) = \int_{\mathcal{Y}} \left[\int_{\Theta} \mathcal{L}[\theta, d(y)] p(y|\theta) \pi(\theta) d\theta \right] dy,$$

where the term in the brackets is posterior expected loss. Minimizing posterior expected loss for every $y \in \mathcal{Y}$, is clearly equivalent to minimizing posterior expected risk, provided it is possible to interchange the order of integration.

The previous definitions did not explicitly state that the prior distribution was proper, that is, that $\int_{\Theta} \pi(\theta) d\theta = 1$. In some applications and for some parameters, researchers may use priors that do not integrate, $\int_{\Theta} \pi(\theta) d\theta = \infty$, commonly called improper priors. A generalized Bayes rule is one that minimizes $r(\pi, d)$, where π is not necessarily a distribution, if such a rule exists. If $r(\pi, d) < \infty$, then the mechanics of this rule is clear, although its meaning is less clear.

4.4 Unintuitive Nature of Decision Making

Despite the elegant mathematical framework of Utility Theory and Statistical Decision Theory established by Savage and von Neumann, actual human decision-making often deviates from these normative axioms. Experiments by **Ellsberg** and **Allais** demonstrated that people frequently violate the axioms of independence and consistency when faced with ambiguity or specific risk profiles.

Example 4.5 (Ellsberg Paradox: Ambiguity Aversion). The Ellsberg paradox is a thought experiment that was first proposed by Daniel Ellsberg in 1961. It is a classic example of a situation where individuals exhibit ambiguity aversion, meaning that they prefer known risks over unknown risks. The paradox highlights the importance of considering ambiguity when making decisions under uncertainty.

There are two urns each containing 100 balls. It is known that urn A contains 50 red and 50 black, but urn B contains an unknown mix of red and black balls. The following bets are offered to a participant:

Bet	Condition	Payoff if True	Payoff if False
1A	Red drawn from urn A	\$1	\$0
2A	Black drawn from urn A	\$1	\$0
1B	Red drawn from urn B	\$1	\$0
2B	Black drawn from urn B	\$1	\$0

Most participants prefer bet 1A over 1B (preferring the known 50% chance over the unknown probability of drawing red from urn B), and they also prefer bet 2A over 2B (again preferring the known 50% chance over the unknown probability of drawing black from urn B).

This pattern of preferences violates the axioms of expected utility theory. If we denote the probability of drawing a red ball from urn B as p , then:

- Preferring 1A over 1B implies: $0.5 > p$
- Preferring 2A over 2B implies: $0.5 > (1 - p)$, which means $p > 0.5$

These two inequalities are contradictory, yet this preference pattern is commonly observed. The paradox demonstrates that people exhibit *ambiguity aversion*: they prefer known probabilities (risk) over unknown probabilities (ambiguity), even when the expected values might be similar. This behavior cannot be explained by standard expected utility theory, which treats all probabilities symmetrically regardless of whether they are known or unknown.

The Ellsberg paradox has important implications for decision-making in real-world situations where probabilities are often ambiguous or unknown, such as in financial markets, insurance, or strategic business decisions. It vividly demonstrates that human decision-making is sensitive not just to risk (known probabilities) but to *ambiguity* (unknown probabilities). Standard expected utility theory fails to capture this “ambiguity aversion,” leading to the development of more generalized decision theories that incorporate confidence in probability estimates.

Example 4.6 (Allais Paradox: Independence Axiom). The Allais paradox is a choice problem designed by Maurice Allais to show an inconsistency of actual observed choices with the predictions of expected utility theory. The paradox is that the choices made in the second problem seem irrational, although they can be explained by the fact that the independence axiom of expected utility theory is violated.

We run two experiments. In each experiment a participant has to make a choice between two gambles.

Experiment 1**Experiment 2**

Gamble G_1		Gamble G_2		Gamble G_3		Gamble G_4	
Win	Chance	Win	Chance	Win	Chance	Win	Chance
\$25m	0	\$25m	0.1	\$25m	0	\$25m	0.1
\$5m	1	\$5m	0.89	\$5m	0.11	\$5m	0
\$0m	0	\$0m	0.01	\$0m	0.89	\$0m	0.9

The difference in expected gains is identical in two experiments

```
E1 <- 5 * 1
E2 <- 25 * 0.1 + 5 * 0.89 + 0 * 0.01
E3 <- 5 * 0.11 + 0 * 0.89
E4 <- 25 * 0.1 + 0 * 0.9
print(c(E1 - E2, E3 - E4))
## -2 -2
```

However, typically a person prefers G_1 to G_2 and G_4 to G_3 , we can conclude that the expected utilities of the preferred are greater than the expected utilities of the second choices. The fact is that if $G_1 \geq G_2$ then $G_3 \geq G_4$ and vice-versa.

Assuming the subjective probabilities $P = (p_1, p_2, p_3)$. The expected utility $E(u|P)$ is $u(0) = 0$ and for the high prize set $u(\$25 \text{ million}) = 1$, which leaves one free parameter $u = u(\$5 \text{ million})$.

Hence to compare gambles with probabilities P and Q we look at the difference

$$E(u|P) - E(u|Q) = (p_2 - q_2)u + (p_3 - q_3)$$

For comparing G_1 and G_2 we get

$$\begin{aligned} E(u|G_1) - E(u|G_2) &= 0.11u - 0.1 \\ E(u|G_3) - E(u|G_4) &= 0.11u - 0.1 \end{aligned}$$

The order is the same, given your u . If your utility satisfies $u < 0.1/0.11 = 0.909$ you take the “riskier” gamble.

The Allais paradox is particularly damaging to the *Independence Axiom*, which states that if you prefer A to B, you should also prefer A+C to B+C. Here, adding a common consequence flips the preference. This finding suggests that people value “certainty” disproportionately, a phenomenon famously captured by Prospect Theory.

Example 4.7 (Winner’s Curse). One of the interesting facts about expectation is that when you are in a competitive auctioning game then you shouldn’t

value things based on pure expected value. You should take into consideration the event that you win W . Really you should be calculating $E(X | W)$ rather than $E(X)$.

The winner's curse: given that you win, you should feel regret: $E(X | W) < E(X)$.

A good example is claiming racehorse whose value is uncertain.

Value	Outcome
0	horse never wins
50,000	horse improves

Simple expected value tells you

$$E(X) = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 50,000 = \$25,000.$$

In a \$20,000 claiming race (you can buy the horse for this fixed fee ahead of time from the owner) it looks like a simple decision to claim the horse.

It's not so simple! We need to calculate a conditional expectation. What's $E(X | W)$, given you win event (W)? This is the expected value of the horse given that you win that is relevant to assessing your bid. In most situations $E(X | W) < 20,000$.

Another related feature of this problem is *asymmetric information*. The owner or trainer of the horse may know something that you don't know. There's a reason why they are entering the horse into a claiming race in the first place.

Winner's curse implies that immediately after you have won, you should feel a little regret, as the object is less valuable to you after you have won! Or put another way, in an auction nobody else in the room is willing to offer more than you at that time.

Example 4.8 (The Hat Problem). There are N prisoners in a forward facing line. Each guy is wearing a blue or red hat. Everyone can see all the hats in front of him, but cannot see his own hat. The hats can be in any combination of red and blue, from all red to all blue and every combination in between. The first guy doesn't know his own hat.

A guard is going to walk down the line, starting in the back, and ask each prisoner what color hat they have on. They can only answer "blue" or "red." If they answer incorrectly, or say anything else, they will be shot dead on the spot. If they answer correctly, they will be set free. Each prisoner can hear all of the other prisoners' responses, as well as any gunshots that indicate an incorrect response. They can remember all of this information.

There is a rule that all can agree to follow such that the first guy makes a choice (“My hat is ...”) and everyone after that, including the last guy, will get their color right with probability 1.

Here is the strategy:

1. The last prisoner (Prisoner 100) counts the number of blue hats among the 99 people in front of him.
2. If he sees an **even** number of blue hats, he yells “Blue”. If **odd**, he yells “Red”. This yell conveys the *parity* of blue hats to everyone else.
3. Prisoner 99 hears the yell. Now knowing the total parity of blue hats (for 1..99), he counts the blue hats he sees (on 1..98).
 - If the total parity (from 100) matches the parity he sees, his own hat must be Red (contributing 0 to the count).
 - If the parities differ, his own hat must be Blue (changing the parity).
4. He yells his calculated color, saving himself and passing the parity information down to Prisoner 98.
5. This induction continues, allowing every prisoner except the first to determine their hat color with certainty. The first prisoner survives with probability 0.5 (random guess that conveys the bit).

One hundred prisoners are too many to work with. Suppose there are two (1 and 2, where 2 is the back). Prisoner 2 sees prisoner 1. If he sees Blue, he yells “Red” (odd). Prisoner 1 hears “Red”, sees 0 blue hats (even). Mismatch -> Blue. This logic extends to any N . By using parity, the group collectively solves the problem with only 1 bit of uncertainty for the entire group.

Example 4.9 (Lemon’s Problem). The lemon problem is an interesting conditional probability puzzle and is a classic example of asymmetric information in economics. It was first proposed by George Akerlof in his 1970 paper “The Market for Lemons: Quality Uncertainty and the Market Mechanism.” The problem highlights the importance of information in markets and how it can lead to adverse selection, where the quality of goods or services is lower than expected.

The basic tenet of the lemons principle is that low-value cars force high-value cars out of the market because of the asymmetrical information available to the buyer and seller of a used car. This is primarily due to the fact that a seller does not know what the true value of a used car is and, therefore, is not willing to pay a premium on the chance that the car might be a lemon. Premium-car sellers are not willing to sell below the premium price so this results in only lemons being sold.

Suppose that a dealer pays \$20K for a car and wants to sell for \$25K. Some cars on the market are Lemons. The dealer knows whether a car is a lemon. A lemon

is only worth \$5K. There is asymmetric information as the customer doesn't know if the particular new car is a lemon. S/he estimates the probability of lemons on the road by using the observed frequency of lemons. We will consider two separate cases:

- Let's first suppose only 10% of cars are lemons.
- We'll then see what happens if 50% are lemons.

The question is how does the market clear (i.e. at what price do car's sell). Or put another way does the customer buy the car and if so what price is agreed on? This is very similar to winner's curse: when computing an expected value what conditioning information should I be taking into account?

In the case where the customer thinks that $p = 0.10$ of the car's are lemons, they are willing to pay

$$E(X) = \frac{9}{10} \cdot 25 + \frac{1}{10} \cdot 5 = \$23K$$

This is greater than the initial \$20 that the dealer paid. The car then sells at \$23K < \$25K.

Of course, the dealer is disappointed that there are lemons on the road as he is not achieving the full value – missing \$2000. Therefore, they should try and persuade the customer its not a lemon by offering a warranty for example.

The more interesting case is when $p = 0.5$. The customer now values the car at

$$E(X) = \frac{1}{2} \cdot 25 + \frac{1}{2} \cdot 5 = \$15K$$

This is lower than the \$20K – the reservation price that the dealer would have for a good car. Now what type of car and at what price do they sell?

The key point in asymmetric information is that the customer must condition on the fact that if the dealer still wants to sell the car, the customer must update his probability of the type of the car. We already know that if the car is not a lemon, the dealer won't sell under his initial cost of \$20K. So at \$15K he is only willing to sell a lemon. But then if the customer computes a conditional expectation $E(X | \text{Lemon})$ – conditioning on new information that the car is a lemon L we get the valuation

$$E(X | L) = 1 \cdot 5 = \$5K$$

Therefore only lemons sell, at \$ 5K, even if the dealer has a perfectly good car the customer is not willing to buy!

Again what should the dealer do? Try to raise the quality and decrease the frequency of lemons in the observable market. This type of modeling has all been used to understand credit markets and rationing in periods of loss of confidence.

Example 4.10 (Envelope Paradox). The envelope paradox is a thought experiment or puzzle related to decision-making under uncertainty. It is also known as the “exchange paradox” or the “two-envelope paradox.” The paradox highlights the importance of carefully considering the information available when making decisions under uncertainty and the potential pitfalls of making assumptions about unknown quantities.

A swami puts m dollars in one envelope and $2m$ in another. He hands one envelope to you and one to your opponent. The amounts are placed randomly and so there is a probability of $\frac{1}{2}$ that you get either envelope.

You open your envelope and find x dollars. Let y be the amount in your opponent’s envelope. You know that $y = \frac{1}{2}x$ or $y = 2x$. You are thinking about whether you should switch your opened envelope for the unopened envelope of your friend. It is tempting to do an expected value calculation as follows

$$E(y) = \frac{1}{2} \cdot \frac{1}{2}x + \frac{1}{2} \cdot 2x = \frac{5}{4}x > x$$

Therefore, it looks as if you should switch no matter what value of x you see. A consequence of this, following the logic of backwards induction, that even if you didn’t open your envelope that you would want to switch! Where’s the flaw in this argument?

This problem permits multiple interpretations, yet it serves as an excellent case study for distinguishing between frequentist and Bayesian reasoning. Rather than addressing every possible condition, we will focus on the most instructive cases. First, assume we are risk-neutral (note that we can simply substitute “money” with “utility” without loss of generality). We will compare frequentist vs. Bayesian, and open vs. closed envelope scenarios.

If I DO NOT look in my envelope, in this case, even from a frequentist viewpoint, we can find a fallacy in this naive expectation reasoning $E[\text{trade}] = 5x/4$. First, the right answer from a frequentist view is, loosely, as follows. If we switch the envelope, we can obtain m (when $X = m$) or lose m (when $X = 2m$) with the same probability $1/2$. Thus, the value of a trade is zero, so that trading matters not for my expected wealth.

Instead, naive reasoning is confusing the property of variables X and m , X is a random variable and m is a fixed parameter which is constant (again, from a frequentist viewpoint). By trading, we can obtain m or lose m with the same probability. Here, the former $X = m$ is different from the latter $X = 2m$. Thus, $X\frac{1}{2} - \frac{X}{2}\frac{1}{2} = \frac{X}{4}$ is the wrong expected value of trading. On the other hand, from a bayesian view, since we have no information, we are indifferent to either trading or not.

The second scenario is if I do look in my envelope. As the Christensen & Utts (1992) article said, the classical view cannot provide a completely reasonable resolution to this case. It is just ignoring the information revealed. Also, the

arbitrary decision rule introduced at the end of the paper or the extension of it commented by Ross (1996) are not the results of reasoning from a classical approach. However, the bayesian approach provides a systematic way of finding an optimal decision rule using the given information.

We can use the Bayes rule to update the probabilities of which envelope your opponent has! Assume $p(m)$ of dollars to be placed in the envelope by the swami. Such an assumption then allows us to calculate an odds ratio

$$\frac{p(Y = \frac{1}{2}x | X = x)}{p(Y = 2x | X = x)}$$

concerning the likelihood of which envelope your opponent has.

Then, the expected value is given by

$$E(Y | X = x) = p\left(Y = \frac{1}{2}x | X = x\right) \cdot \frac{1}{2}x + p(Y = 2x | X = x) \cdot 2x$$

and the condition $E(Y | X = x) > x$ becomes a decision rule.

Let $g(m)$ be the prior distribution of m . Applying Bayes' theorem, we have

$$p(m = x | X = x) = \frac{p(X = x | m = x)g(x)}{p(X = x)} = \frac{g(x)}{g(x) + g(x/2)}.$$

Similarly, we have

$$p(m = x/2 | X = x) = \frac{p(X = x | m = x/2)g(x/2)}{p(X = x/2)} = \frac{g(x/2)}{g(x) + g(x/2)}.$$

The Bayesian can now compute his expected winnings from the two actions. If he keeps the envelope he has, he wins x dollars. If he trades envelopes, he wins $x/2$ if he currently has the envelope with $2m$ dollars, i.e., if $m = x/2$ and he wins $2x$ if he currently has the envelope with m dollars, i.e., $m = x$. His expected winnings from a trade are

$$E(W | Trade) = E(Y | X = x) = \frac{g(x/2)}{g(x) + g(x/2)} \frac{x}{2} + \frac{g(x)}{g(x) + g(x/2)} 2x.$$

It is easily seen that when $g(x/2) = 2g(x)$, $E(W | Trade) = x$. Therefore, if $g(x/2) > 2g(x)$ it is optimal to keep the envelope and if $g(x/2) < 2g(x)$ it is optimal to trade envelopes. For example, if your prior distribution on m is exponential λ , so that $g(m) = \lambda e^{-\lambda m}$, then it is easily seen that it is optimal to keep your envelope if $x > 2 \log(2)/\lambda$.

The intuitive value of the expected winnings when trading envelopes was shown to be $5x/4$. This value can be obtained by assuming that $g(x)/[g(x) + g(x/2)] = 1/2$ for all x . In particular, this implies that $g(x) = g(x/2)$ for all x , i.e., $g(x)$ is a constant function. In other words, the intuitive expected

winnings assumes an improper “noninformative” uniform density on $[0, \infty)$. It is of interest to note that the improper noninformative prior for this problem gives a truly noninformative (maximum entropy) posterior distribution.

Most of the arguments in the Christensen & Utts (1992) paper are right, but there is one serious error in the article which is corrected in Bachman-Christensen-Utts (1996) and discussed in Brams & Kilgour (1995). The paper calculated the marginal density of X like below.

$$\begin{aligned} p(X = x) &= p(m = x)g(x) + p(2m = x)g(x/2) \\ &= \frac{1}{2}g(x) + \frac{1}{2}g(x/2) \end{aligned}$$

where $g(x)$ is the prior distribution of m . However, integrating $p(X = x)$ with respect to x from 0 to ∞ gives 3/2 instead of 1. In fact, their calculation of $p(X = x)$ can hold only when the prior distribution $g(x)$ is discrete and $p(X = x)$, $g(m)$, $g(m/2)$ represent the probabilities that $X = x$, $m = m$, $m = m/2$, respectively.

For the correct calculation of the continuous X case, one needs to properly transform the distribution. That can be done by remembering to include the Jacobian term alongside the transformed PDF, or by working with the CDF of X instead. The latter forces one to properly consider the transform, and we proceed with that method.

Let $G(x)$ be the CDF of the prior distribution of m corresponding to $g(x)$.

$$\begin{aligned} p(x < X \leq x + dx) &= p(m = x)dG(x) + p(2m = x)dG(x/2) \\ &= \frac{1}{2}(dG(x) + dG(x/2)) \end{aligned}$$

where $g(x) = dG(x)/dx$. Now, the PDF of X is

$$\begin{aligned} f_X(x) &= \frac{d}{dx}p(x < X \leq x + dx) \\ &= \frac{1}{2}\left(g(x) + \frac{1}{2}g(x/2)\right) \end{aligned}$$

We have an additional 1/2 in the last term due to the chain rule, or the Jacobian in the change-in-variable formula. (Recall that when transforming a probability density $f_X(x)$ to $f_Y(y)$ where $y = g(x)$, we must scale by $|dx/dy|$ to preserve the total probability mass of 1). Therefore, the expected amount of a trade is

$$\begin{aligned} E(Y | X = x) &= \frac{x}{2}p(2m = x | X = x) + 2x p(m = x | X = x) \\ &= \frac{x}{2} \frac{g(x)}{g(x) + g(x/2)/2} + 2x \frac{g(x/2)/2}{g(x) + g(x/2)/2} \\ &= \frac{\frac{x}{2}g(x) + xg(x/2)}{g(x) + g(x/2)/2} \end{aligned}$$

Thus, for the continuous case, trading is advantageous whenever $g(x/2) < 4g(x)$, instead of the decision rule for the discrete case $g(x/2) < 2g(x)$.

Now, think about which prior will give you the same decision rule as the frequentist result. In the discrete case, $g(x)$ such that $g(x/2) = 2g(x)$, and in the continuous case $g(x)$ such that $g(x/2) = 4g(x)$. However, both do not look like useful, non-informative priors. Therefore, the frequentist approach does not always equal the Bayes approach with a non-informative prior. At the moment you start to treat x as a given number, and consider $p(m | X = x)$ (or $p(Y | X = x)$), you are thinking in a bayesian way, and need to understand the implications and assumptions in that context.

4.5 Decision Trees

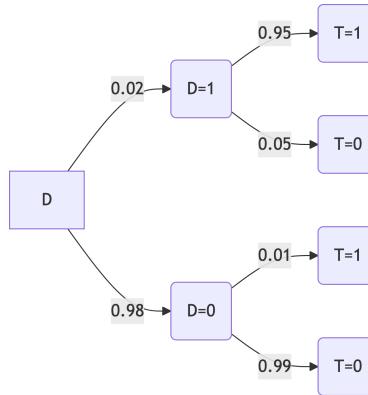
Decision trees can effectively model and visualize conditional probabilities. They provide a structured way to break down complex scenarios into smaller, more manageable steps, allowing for clear calculations and interpretations of conditional probabilities.

Each node in a decision tree, including the root, represents an event or condition. The branches represent the possible outcomes of that condition. Along each branch, you'll often see a probability. This is the chance of that outcome happening, given the condition at the node. As you move down the tree, you're looking at more specific conditions and their probabilities. The leaves of the tree show the final probabilities of various outcomes, considering all the conditions along the path to that leaf. Thus, the probabilities of the leaves need to sum to 1.

Example 4.11 (Medical Testing). A patient goes to see a doctor. The doctor performs a test which is 95% sensitive – that is 95 percent of people who are sick test positive and 99% specific – that is 99 percent of the healthy people test negative. The doctor also knows that only 2 percent of the people in the country are sick. Now the question is: if the patient tests positive, what are the chances the patient is sick? The intuitive answer is 99 percent, but the correct answer is 66 percent.

Formally, we have two binary variables, $D = 1$ that indicates you have a disease and $T = 1$ that indicates that you test positive for it. The estimates we know already are given by $P(D) = 0.02$, $P(T | D) = 0.95$, and $P(\bar{T} | \bar{D}) = 0.99$. Here we used shortcut notations, instead of writing $P(D = 1)$ we used $P(D)$ and instead of $P(D = 0)$ we wrote $P(\bar{D})$.

Sometimes it is more intuitive to describe probabilities using a tree rather than tables. The tree below shows the conditional distribution of D and T .



The result is counter-intuitive. Let's think about this intuitively. Rather than relying on Bayesian math to help us with this, let us consider another illustration. Imagine that the above story takes place in a small town, with 1,000 people. From the prior $P(D) = 0.02$, we know that 2 percent, or 20 people, are sick, and 980 are healthy. If we administer the test to everyone, the most probable result is that 19 of the 20 sick people test positive. Since the test has a 1 percent error rate, however, it is also probable that 9.8 of the healthy people test positive, we round it to 10.

Now if the doctor sends everyone who tests positive to the national hospital, there will be 10 healthy and 19 sick patients. If you meet one, even though you are armed with the information that the patient tested positive, there is only a 66 percent chance this person is sick.

Let's extend the example and add the utility of the test and the utility of the treatment. Then the decision problem is to treat a_T or not to treat a_N . The Q-function is the function of the state $S \in \{D_0, D_1\}$ and the action $A \in \{a_T, a_N\}$

Table 4.8: Utility of the test and the treatment.

A/S	a_T	a_N
D_0	90	100
D_1	90	0

Then expected utility of the treatment is 90 and no treatment is 98. A huge difference. Given our prior knowledge, we should not treat everyone.

```

0.02 * 90 + 0.98 * 90 # treat
## 90
  
```

```
0.02 * 0 + (1 - 0.02) * 100 # do not treat
## 98
```

However, the expected utility will change when our probability of disease changes. Let's say that we are in a country where the probability of disease is 0.1 or we performed a test and updated our prior probability of disease to some number p . Then the expected utility of the treatment is $E[U(a_T)] = 90$ and no treatment is

$$E[U(a_N)] = 0 \cdot p + 100 \cdot (1 - p) = 100(1 - p)$$

When we are unsure about the value of p we may want to explore how the optimal decision changes as we vary p

```
p <- seq(0, 1, 0.01)
plot(p, 100 * (1 - p), type = "l", xlab = "p", ylab = "E[U(a)]")
abline(h = 90, col = "red")
legend("bottomleft", legend = c(
  TeX("$E[U(a_N)]$"),
  TeX("$E[U(a_T)]$"))
, col = c("black", "red"), lty = 1, bty = "n")
```

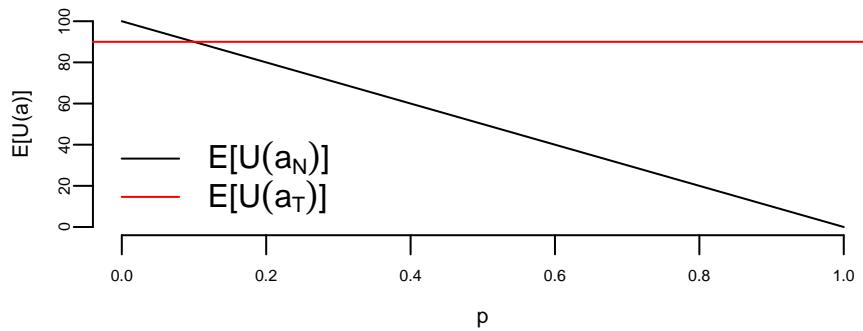


Figure 4.2: Expected utility of the treatment and no treatment as a function of the prior probability of disease.

If our estimate is at the crossover point, then we should be indifferent between treatment and no treatment, if on the left of the crossover point, we should treat, and if on the right, we should not treat. The crossover point is.

$$100(1 - p) = 90, p = 0.1$$

The gap of $90 - 100(1 - p)$ is the expected gain from treatment.

Now, let us calculate the value of test, e.g. the change in expected utility from the test. We will need to calculate the posterior probabilities

```
# P(D | T = 0) = P(T = 0 | D) P(D) / P(T = 0)
pdt0 <- 0.05 * 0.02 / (0.05 * 0.02 + 0.99 * 0.98)
# Expected utility given the test is negative
# E[U(a_N | T=0)]
UN0 <- pdt0 * 0 + (1 - pdt0) * 100
# E[U(a_T | T=0)]
UTO <- pdt0 * 90 + (1 - pdt0) * 90

sprintf("P(D | T = 0) : %.4f, E[U(a_N | T=0)] : %.1f, E[U(a_T |
    ↵ T=0)] : %.1f", pdt0, UN0, UTO)
## "P(D | T = 0) : 0.0010, E[U(a_N | T=0)] : 99.9, E[U(a_T |
    ↵ T=0)] : 90.0"
```

Given test is negative, our best action is not to treat. Our utility is 100. What if the test is positive?

```
# P(D | T = 1) = P(T = 1 | D) P(D) / P(T = 1)
pdt <- 0.95 * 0.02 / (0.95 * 0.02 + 0.01 * 0.98)
# E[U(a_N | T=1)]
UN1 <- pdt * 0 + (1 - pdt) * 100
# E[U(a_T | T=1)]
UT1 <- pdt * 90 + (1 - pdt) * 90

## "P(D | T = 1) : 0.6597, E[U(a_N | T=1)] : 34.0, E[U(a_T | T=1)] : 90.0"
```

The best option is to treat now! Given the test our strategy is to treat if the test is positive and not treat if the test is negative. Let's calculate the expected utility of this strategy.

```
# P(T=1) = P(T=1 | D) P(D) + P(T=1 | D=0) P(D=0)
pt <- 0.95 * 0.02 + 0.01 * 0.98
# P(T=0) = P(T=0 | D) P(D) + P(T=0 | D=0) P(D=0)
pt0 <- 0.05 * 0.02 + 0.99 * 0.98
# Expected utility of the strategy
sprintf("P(T=1) : %.4f, P(T=0) : %.4f, E[U(a)] : %.1f", pt, pt0,
    ↵ pt * UT1 + pt0 * UN0)
## "P(T=1) : 0.0288, P(T=0) : 0.9712, E[U(a)] : 99.6"
```

The utility of our strategy of 100 is above the strategy prior to testing (98), this difference of 2 is called the *value of information*.

Example 4.12 (Mudslide). I live in a house that is at risk of being damaged by a mudslide. I can build a wall to protect it. The wall costs \$10,000. If there is a mudslide, the wall will protect the house with probability 0.95. If there is no mudslide, the wall will not cause any damage. The prior probability of a mudslide is 0.01. If there is a mudslide and the wall does not protect the house, the damage will cost \$100,000. Should I build the wall?

Let's formally solve this as follows:

- Build a decision tree.
- The tree will list the probabilities at each node. It will also list any costs there are you going down a particular branch.
- Finally, it will list the expected cost of going down each branch, so we can see which one has the better risk/reward characteristics.

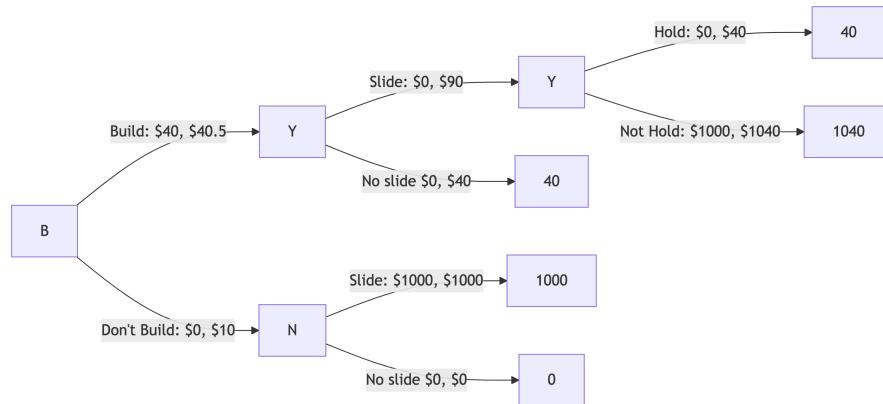


Figure 4.3: Mudslide Decision Tree

The first dollar value is the cost of the edge, e.g. the cost of building the wall is \$10,000. The second dollar value is the expected cost of going down that branch. For example, if you build the wall and there is a mudslide, the expected cost is \$15,000. If you build the wall and there is no mudslide, the expected cost is \$10,000. The expected cost of building the wall is \$10,050. The expected cost of not building the wall is \$1,000. The expected cost of building the wall is greater than the expected cost of not building the wall, so you should not build the wall. The dollar value at the leaf nodes is the expected cost of going down that branch. For example, if you build the wall and there is a mudslide and the wall does not hold, the expected cost is \$110,000.

There's also the possibility of a further test to see if the wall will hold. Let's include the geological testing option. The test costs \$3000 and has the following

accuracies.

$$P(T \mid \text{Slide}) = 0.90 \text{ and } P(\text{not } T \mid \text{No Slide}) = 0.85$$

If you choose the test, then should you build the wall?

Let's use the Bayes rule. The initial prior probabilities are

$$P(\text{Slide}) = 0.01 \text{ and } P(\text{No Slide}) = 0.99$$

$$\begin{aligned} P(T) &= P(T \mid \text{Slide})P(\text{Slide}) + P(T \mid \text{No Slide})P(\text{No Slide}) \\ P(T) &= 0.90 \times 0.01 + 0.15 \times 0.99 = 0.1575 \end{aligned}$$

We'll use this to find our optimal course of action.

The posterior probability given a positive test is

$$\begin{aligned} P(\text{Slide} \mid T) &= \frac{P(T \mid \text{Slide})P(\text{Slide})}{P(T)} \\ &= \frac{0.90 \times 0.01}{0.1575} = 0.0571 \end{aligned}$$

The posterior probability given a negative test is

$$\begin{aligned} P(\text{Slide} \mid \text{not } T) &= \frac{P(\text{not } T \mid \text{Slide})P(\text{Slide})}{P(\text{not } T)} \\ &= \frac{0.1 \times 0.01}{0.8425} \\ &= 0.001187 \end{aligned}$$

Compare this to the initial base rate of a 1% chance of having a mud slide.

Given that you build the wall without testing, what is the probability that you'll lose everything? With the given situation, there is one path (or sequence of events and decisions) that leads to losing everything:

- Build without testing (given) Slide (0.01)
- Doesn't hold (0.05)

$$P(\text{losing everything} \mid \text{build w/o testing}) = 0.01 \times 0.05 = 0.0005$$

Given that you choose the test, what is the probability that you'll lose everything? There are two paths that lead to losing everything:

- There are three things that have to happen to lose everything. Test +ve ($P = 0.1575$), Build, Slide ($P = 0.0571$), Doesn't Hold ($P = 0.05$)
- Now you lose everything if Test -ve ($P = 0.8425$), Don't Build, Slide given negative ($P = 0.001187$).

The conditional probabilities for the first path

$$P(\text{first path}) = 0.1575 \times 0.0571 \times 0.05 = 0.00045$$

For the second path

$$P(\text{second path}) = 0.8425 \times 0.001187 = 0.00101$$

Hence putting it all together

$$P(\text{losing everything} \mid \text{testing}) = 0.00045 + 0.00101 = 0.00146$$

Putting these three cases together we can build a risk/reward table

Choice	Expected Cost	Risk	P
Don't Build	\$1,000	0.01	1 in 100
Build w/o testing	\$10,050	0.0005	1 in 2000
Test	\$4,693	0.00146	1 in 700

The expected cost with the test is $3000 + 10000 \times 0.1575 + 100000 \times 0.001187 = 4693$

What do you choose?

4.6 Nash Equilibrium

When multiple decision makers interact with each other, meaning the decision of one player changes the state of the “world” and thus affects the decision of another player, then we need to consider the notion of equilibrium. It is a central concept in economics and game theory. The most widely used type of equilibrium is the Nash equilibrium, named after John Nash, who introduced it in his 1950 paper “Equilibrium Points in N-Person Games.” It was popularized by the 1994 film “A Beautiful Mind,” which depicted Nash’s life and work.

It is defined as a set of strategies where no player can improve their payoff by unilaterally changing their strategy, assuming others keep their strategies constant. In other words, a Nash equilibrium is a set of strategies where no player has an incentive to deviate from their current strategy, given the strategies of the other players.

Here are a few examples of Nash equilibria:

- Prisoner's Dilemma: Two prisoners must decide whether to cooperate with each other or defect. The Nash equilibrium is for both to defect, even though they would be better off if they both cooperated.
- Pricing Strategies: Firms in a market choose prices to maximize profits, taking into account their competitors' pricing decisions. The equilibrium is the set of prices where no firm can increase profits by changing its price unilaterally.
- Traffic Flow: Drivers choose routes to minimize travel time, based on their expectations of other drivers' choices. The equilibrium is the pattern of traffic flow where no driver can reduce their travel time by choosing a different route.

Example 4.13 (Marble Game). Here is a subtle marble game where players have to call out (or present) either red or blue with different payoffs according to how things match. Two players *A* and *B* have both a red and a blue marble. They present one marble to each other. The payoff table is as follows:

- If both present red, *A* wins \$3.
- If both present blue, *A* wins \$1.
- If the colors do not match, *B* wins \$2

The question is whether it is better to be *A* or *B* or does it matter? Moreover, what kind of strategy should you play? A lot depends on how much credit you give your opponent. A lot of empirical research was done on the *tit-for-tat* strategy, where you cooperate until your opponent defects. Then you match his last response.

Nash equilibrium will also allow us to study the concept of a *randomized strategy* (ie. picking a choice with a certain probability) which turns out to be optimal in many game theory problems.

First, assume that the players have a $\frac{1}{2}$ probability of playing Red or Blue. Thus each player has the same expected payoff $E(A) = \$1$

$$E(A) = \frac{1}{4} \cdot 3 + \frac{1}{4} \cdot 1 = 1$$

$$E(B) = \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 2 = 1$$

We might go one step further and look at the risk (and measured by a standard deviation) and calculate the variances of each player's payouts.

$$\begin{aligned} \text{Var}(A) &= (1 - 1)^2 \cdot \frac{1}{4} + (3 - 1)^2 \cdot \frac{1}{4} + (0 - 1)^2 \cdot \frac{1}{2} = 1.5 \\ \text{Var}(B) &= 1^2 \cdot \frac{1}{2} + (2 - 1)^2 \cdot \frac{1}{2} = 1 \end{aligned}$$

Therefore, under this scenario, if you are risk averse, player B position is favored.

The matrix of probabilities with equally likely choices is given by

A, B	Probability
$P(\text{red}, \text{red})$	$(1/2)(1/2)=1/4$
$P(\text{red}, \text{blue})$	$(1/2)(1/2)=1/4$
$P(\text{blue}, \text{red})$	$(1/2)(1/2)=1/4$
$P(\text{blue}, \text{blue})$	$(1/2)(1/2)=1/4$

Now there is no reason to assume ahead of time that the players will decide to play 50/50. We will show that there's a mixed strategy (randomized) that is a *Nash equilibrium* that is, both players won't deviate from the strategy. We'll prove that the following equilibrium happens:

- A plays Red with probability 1/2 and blue 1/2
- B plays Red with probability 1/4 and blue 3/4

In this case the expected payoff to playing Red equals that of playing Blue for each player. We can simply calculate: A 's expected payoff is 3/4 and B 's is \$1

$$E(A) = \frac{1}{8} \cdot 3 + \frac{3}{8} \cdot 1 = \frac{3}{4}$$

Moreover, $E(B) = 1$, thus $E(B) > E(A)$. We see that B is the favored position. It is clear that if I know that you are going to play this strategy and vice-versa, neither of us will deviate from this strategy – hence the Nash equilibrium concept.

Nash equilibrium probabilities are: $p = P(A \text{ red}) = 1/2$, $p_1 = P(B \text{ red}) = 1/4$ with payout matrix

A, B	Probability
$P(\text{red}, \text{red})$	$(1/2)(1/4)=1/8$
$P(\text{red}, \text{blue})$	$(1/2)(3/4)=3/8$

A, B	Probability
$P(\text{blue, red})$	$(1/2)(1/4)=1/8$
$P(\text{blue, blue})$	$(1/2)(3/4)=3/8$

We have general payoff probabilities: $p = P(A \text{ red})$, $p_1 = P(B \text{ red})$

$$\begin{aligned} f_A(p, p_1) &= 3pp_1 + (1-p)(1-p_1) \\ f_B(p, p_1) &= 2\{p(1-p_1) + (1-p)p_1\} \end{aligned}$$

To find the equilibrium point

$$\begin{aligned} (\partial/\partial p)f_A(p, p_1) &= 3p_1 - (1-p_1) = 4p_1 - 1 \text{ so } p_1 = 1/4 \\ (\partial/\partial p_1)f_B(p, p_1) &= 2(1-2p) \text{ so } p = 1/2 \end{aligned}$$

Much research has been directed to repeated games versus the one-shot game and is too large a topic to discuss further. However, we can analyze one particularly famous strategy: tit-for-tat.

In a repeated version of the marble game (Camerer 2003), player A can employ the tit-for-tat strategy: start by playing Red in the first round, and thereafter simply repeat the opponent's previous move. This strategy was famously shown by Robert Axelrod in his computer tournaments to be remarkably effective in iterated prisoner's dilemma games, despite its simplicity.

Let us analyze how this strategy performs against different opponent behaviors. Consider player A using tit-for-tat against various strategies employed by B .

If B always plays Red, the game locks into a mutually beneficial pattern: both players present Red every round, and A wins \$3 each time. The per-round expected payoff for A is simply \$3.

If B always plays Blue, the dynamics are more interesting. In round 1, A plays Red while B plays Blue, so B wins \$2. From round 2 onward, A copies B 's Blue, resulting in both playing Blue and A winning \$1 per round. Over n rounds, A 's total payoff is $-2 + (n-1) \cdot 1 = n - 3$, giving an average of $(n-3)/n$ which approaches \$1 as $n \rightarrow \infty$.

If both players use tit-for-tat starting with Red, they remain synchronized forever, each playing Red in every round, yielding \$3 per round for A . This mutual cooperation is stable because neither player ever defects.

The more challenging case arises when B employs an alternating strategy. Suppose B plays Red, Blue, Red, Blue, and so on. Then the sequence unfolds as follows: in round 1, $(A, B) = (R, R)$ so A wins \$3; in round 2, A copies B 's Red while B switches to Blue, giving (R, B) and B wins \$2; in round 3, A

copies Blue while B plays Red, giving (B, R) and B wins \$2; this pattern of B winning \$2 continues indefinitely. After the first round, tit-for-tat becomes permanently out of phase with the alternating opponent, resulting in perpetual losses for A .

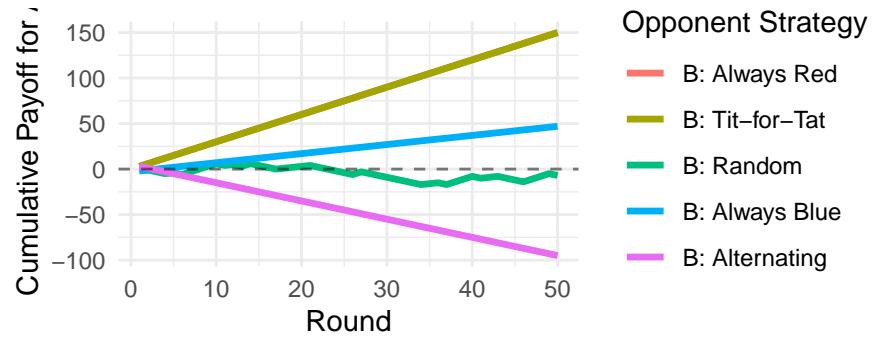


Figure 4.4: Cumulative payoff for player A using tit-for-tat against different opponent strategies over 50 rounds.

The simulation reveals that tit-for-tat performs excellently against cooperative or consistent opponents (always Red, or another tit-for-tat player), reasonably well against random play, but poorly against the alternating strategy which exploits the one-round lag inherent in the copying mechanism.

Table 4.12: Average payoff per round (last 20 rounds)

Opponent	Average Payoff
Always Red	7.50
Tit-for-Tat	7.50
Random	-0.35
Always Blue	2.35
Alternating	-4.75

Average payoff per round (last 20 rounds)

The key insight from Axelrod's tournaments was that tit-for-tat succeeds not by exploiting opponents, but by fostering cooperation. It is *nice* (never defects first), *retaliatory* (punishes defection immediately), *forgiving* (returns to cooperation if the opponent does), and *clear* (easy for opponents to understand and predict). These properties make it robust in environments where opponents can learn and adapt, even though it can be exploited by adversaries who know the strategy and deliberately play to keep it out of phase.

Thomas Schelling, in his influential essay *The Manipulation of Risk*, extends these game-theoretic insights to international relations, arguing that conflicts between major powers often resemble what he calls *competitions in risk taking* or *tests of nerve* rather than direct tests of force. In this framing, the marble game captures something essential about diplomacy: outcomes are determined not by who can bring the most force to bear, but by who is willing to escalate further or who can make it appear that escalation is forthcoming.

Schelling emphasizes the central role of *face* in such interactions, though he uses the term carefully. Face is not mere pride or status; it is a country's (or player's) *reputation for action*, the expectations others hold about how it will behave in future encounters. As Schelling puts it, face is the interdependence of a country's commitments. When player *A* in our marble game employs tit-for-tat and retaliates against defection, *A* is not merely responding to the current round but preserving expectations about future behavior. If *A* were to let defections pass unpunished, *B* would rationally update beliefs about what *A* will tolerate, potentially inviting further exploitation.

This perspective illuminates why tit-for-tat's *retaliatory* property is so important: it maintains credibility. Schelling notes that it would be hard to persuade an adversary, if one yielded repeatedly on minor issues, that one would stand firm on a vital issue. The same logic applies in repeated games: a player who fails to punish defection signals weakness, inviting further defection. Yet Schelling also cautions that face should not attach itself to unworthy enterprises, and that it is sometimes wise to help an adversary save face by providing an exit that does not appear to be capitulation. In game-theoretic terms, this suggests that optimal strategies in repeated interactions must balance retaliation with opportunities for returning to cooperation, precisely the *forgiving* quality that makes tit-for-tat effective.

Equilibrium analysis helps predict the likely outcomes of strategic interactions, even when individuals are acting in their own self-interest. Further, we can use it to understand how markets function and how firms make pricing and production decisions or to design mechanisms (e.g., auctions, voting systems) that incentivize desired behavior and achieve efficient outcomes.

One major drawback is that equilibrium analysis relies on assumptions about rationality and common knowledge of preferences and strategies, which may not always hold in real-world situations. Furthermore, some games may have multiple equilibria, making it difficult to predict which one will be reached. The problem of dynamic strategies, when individuals may learn and adjust their strategies as they gain experience, is hard.

5

A/B Testing

The Internet age opened the door to enormous data collection on personal preferences, behaviors, and actions. The data collected are observational rather than data collected from designed experiments—where we can control the environment to find the effects of interventions. User interface design is a prime example of this. Companies like Google, Amazon, and Netflix run thousands of experiments daily to optimize user experience and revenue. A/B testing is essentially a randomized controlled trial at scale, where two variants (A and B) are compared to determine which one performs better on a specific metric. The statistical machinery remains the same—we formulate a null hypothesis (no difference between variants), collect data, and compute a statistic to see if the observed difference is significant or just noise. The A/B variations can be physical, such as different colors or layouts, or they can be algorithmic, such as different search algorithms or recommendation systems.

However, a key distinction exists between traditional scientific hypothesis testing and modern A/B testing: the objective. In scientific research, the primary goal is often *truth discovery*—establishing a reproducible fact about the world. This necessitates a conservative approach with strict control over False Positives (Type I errors). In contrast, the goal of A/B testing in industry is often *decision making*—choosing the best option to maximize a business metric like revenue or engagement. Here, the cost of a missed opportunity (Type II error) can be just as detrimental as a false alarm. Consequently, industry practitioners may focus more on the magnitude of the effect (effect size) and the expected value of the decision, rather than relying solely on a rigid p-value threshold.

Throughout this chapter we use the classical testing language as an operational planning tool: Type I and Type II errors, power, and significance levels. In particular, α denotes a frequentist Type I error rate (test size). Chapter 6 revisits the same problems from a Bayesian decision-theoretic viewpoint, where posterior probabilities, Bayes factors, and explicit losses take center stage; when we discuss credibility or posterior uncertainty, we will avoid overloading α with a second meaning.

How do we know if the difference between variants is real or just noise? Consider a coin flip: if it comes up heads twice in a row, is the coin biased? Obviously two tosses aren't enough to tell. Hypothesis testing formalizes this

intuition—it tells us whether we have enough evidence to draw a conclusion, or whether we need more data. Let's work through a concrete example.

You work as a quant for a trading firm and you have developed a new algorithm to trade stocks. You tested your algorithm on historical data and it outperformed the state-of-the-art algorithm used in your company. Now, the important question is whether your trading strategy can truly outperform the market or it just got lucky. We need to analyze the performance of the algorithm after it was created and decide whether we have truly discovered a dominant strategy. The effect we try to measure is usually present in some statistics that we calculate from data, for example, sample mean, proportion, or difference in means.

5.1 Hypothesis Testing

Example 5.1 (Pyx Trial). The “Pyx Trial” refers to an ancient ceremony held in the United Kingdom’s Royal Mint. This tradition, dating back to the 12th century, is a method of testing the quality of minted coins to ensure it meets the standards of weight and purity set by law. The term “Pyx” comes from the Greek word “pyxis,” meaning a small box, which is used to hold the sample coins that are to be tested.

Sir Isaac Newton became Warden of the Mint in 1696 and later the Master of the Mint. His role was crucial in reforming the coinage and improving its quality. Newton was rigorous in enforcing standards and combating counterfeiting and clipping (the practice of shaving off small amounts of precious metal from coins). Newton applied his scientific mind to the problems of minting, including refining assays (the testing of the purity of metals), improving the design of coins to prevent clipping, and introducing milled edges on coins.

The trial starts by selecting n coins from each batch produced by the Royal Mint. These coins are placed in a box called the Pyx. The number of coins used in the Trial of the Pyx can vary each year. This number depends on several factors, including the variety and quantity of coins produced by the Royal Mint in that particular year. Typically, a representative sample of each type of coin minted is selected for testing. Then, for each coin attribute (weight, size, composition), the mean (average) value of the sample is calculated as well as the variance of the mean.

Suppose we have minted one million of coins and collected the sample of $n = 100$ coins, the legal weight tolerance for a certain coin is 0.05 grams.

We will use the simulated data for our analysis. Let's simulate the weights of all of the coins produced

Now, we survey 100 randomly selected coins

```
xbar <- mean(survey_sample)
```

The sample mean of 4.9950764 is very close to the true mean of 5. However, if we were to collect a different sample, the sample mean would be slightly different

```
xbar <- mean(survey_sample)
```

Now, we simulate 2000 surveys and calculate the sample mean for each survey.

```
hist(prep, breaks = 30, freq = F, main = "", col = "lightblue")
p <- seq(4.9, 5.1, length.out = 500)
lines(p, dnorm(p, mean(prep), sd(prep)), col = "red", lwd = 3)
```

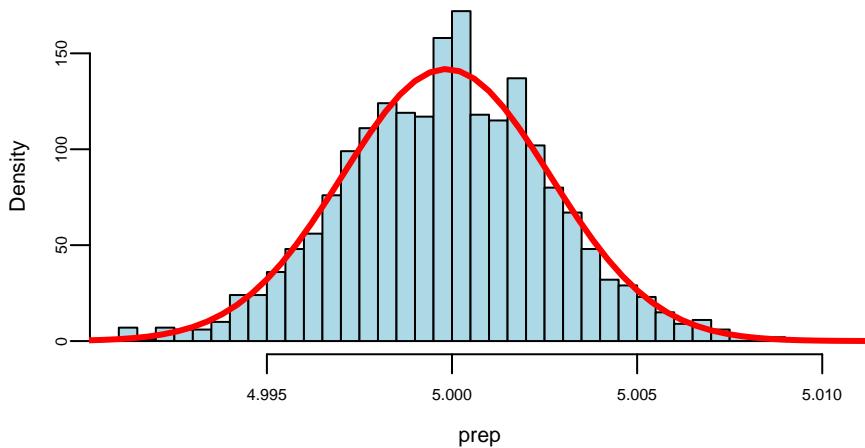


Figure 5.1: Histogram of voting proportions

We see that the red bell-curve (normal density) is a good model for the distribution over the means calculated from samples. In fact, the *central limit theorem* says that sample means follow a normal distribution. We need to estimate the mean and standard deviation of this bell curve. It is natural to use the sample mean as the estimate of the mean of the bell curve `mean(prep)`: 4.9998448. The mean

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

is close to the true population mean of 5. We sometimes use the notation $\hat{\mu}$ to denote an estimate. So we have $\hat{\mu} = \bar{x}$.

However, the standard deviation is much lower compared to the standard deviation of the population

```
sd(prep)
## 0.0028
sd(allcoins)
## 0.029
```

The variance of the mean measures how much the sample mean is expected to vary from one sample to another, if you were to take multiple samples from the same population.

Assuming that samples are uncorrelated (correct sampling procedure is important!), the variance of the mean is given by

$$\text{Var}(\bar{X}) = \text{Var}\left(\frac{1}{n} \sum_{i=1}^n X_i\right) = \frac{1}{n^2} \sum_{i=1}^n \text{Var}(X_i) = \frac{\sigma^2}{n}.$$

Therefore, the variance of the mean formula is

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{n}.$$

If we know the population variance $\text{Var}(X_i) = \sigma^2$, then we can calculate the variance of the mean. However, in practice, we do not know the population variance. Instead, we estimate it using the sample variance s^2 . The estimated variance of the mean is then

$$\text{Var}(\bar{X}) = \frac{s^2}{n}.$$

The standard deviation of the mean is called the *standard error* and is given by

$$s_{\bar{X}} = \sqrt{\text{Var}(\bar{X})} = \frac{s}{\sqrt{n}}.$$

Let's compare the standard error of the mean and standard deviation calculated from the simulations

```
sd(prep)
## 0.0028
sd(allcoins) / sqrt(100)
## 0.0029
```

They are very close!

This statistical property allows us to quantify uncertainty about the sample mean and say something about the true value of the mean μ , in terms of a probabilistic interval statement.

Central Limit Theorem

CLT states that, given a sufficiently large sample size, the distribution of the sample means will be approximately normally distributed, regardless of the shape of the population distribution. This normal distribution is also known as the Gaussian distribution. The theorem applies to a wide range of population distributions, including distributions that are not normal. This universality makes it one of the most powerful and widely-used theorems in statistics.

The Central Limit Theorem originates from the [De Moivre-Laplace theorem](#), published by de Moivre in 1738, which established the normal approximation to the binomial distribution. According to this theorem the standard normal distribution arises as the limit of scaled and centered Binomial distributions, in the following sense. Let x_1, \dots, x_n be independent, identically distributed Rademacher random variables, that is, independent random variables with distribution

$$P(X_i = 1) = P(X_i = -1) = \frac{1}{2}.$$

Then, the distribution of the sum of these random variables converges to the standard normal distribution as n tends to infinity. That is, for any $a < b$, we have

$$\lim_{n \rightarrow \infty} P\left(a \leq \frac{X_1 + \dots + X_n}{\sqrt{n}} \leq b\right) = \int_a^b \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx.$$

In this case, the sum $X_1 + \dots + X_n$ has mean $n\mu$ and variance $n\sigma^2$, so that the standardized sum $(X_1 + \dots + X_n - n\mu)/\sqrt{n\sigma^2}$ has mean 0 and variance 1. The theorem then states that the distribution of this standardized sum converges to the standard normal distribution as n tends to infinity.

In 1889 Francis Galton published a paper where he described what we now call the Galton Board. The Galton Board is a vertical board with interleaved rows of pins. Balls are dropped from the top, and bounce left and right as they hit the pins. Eventually, they are collected into one of several bins at the bottom. The distribution of balls in the bins approximates the normal distribution. Each pin is a physical realization of the binomial draw and each row is a summand. The location at the bottom is a sum of the binomial draws. The `galton-ball.r` script simulates the Galton board experiment. The script is available in the R folder of the book repository. Figure 5.2 shows the result of the simulation. The distribution of the balls in the bins approximates the normal distribution.

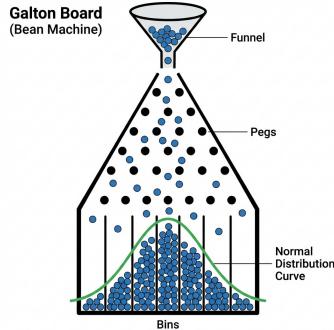


Figure 5.2: Galton Board

Example 5.2 (Android Earthquake Alerts). A fantastic modern example of the Central Limit Theorem in action is Google’s Android Earthquake Alerts System. The core idea is to use the accelerometers present in billions of Android smartphones to create a global earthquake detection network.

Each individual phone’s accelerometer is a “noisy sensor.” It can be triggered by many events that are not earthquakes, such as the phone being dropped, vibrations from a passing truck, or even loud music. A signal from a single phone is therefore a “weak signal” — on its own, it’s not reliable enough to issue an earthquake alert.

However, Google’s system aggregates signals from a vast number of phones in a specific geographic area. When a real earthquake occurs, thousands or even millions of phones in the affected region will detect the seismic waves (P-waves and S-waves) at roughly the same time.

By averaging the readings from this massive number of sensors, the system can effectively cancel out the random noise from individual phones. A single phone dropping is a random, isolated event. But thousands of phones vibrating in a synchronized pattern is a clear, strong signal that is highly unlikely to be due to chance.

This is a direct application of the principle that the standard error of the mean is inversely proportional to the square root of the sample size ($s_{\bar{x}} = s/\sqrt{n}$). Here, n is the number of phones. With a massive n , the standard error of the average measurement becomes incredibly small. This allows the system to have a very high level of confidence that the detected event is a real earthquake, enabling it to send out timely alerts to people who may be in danger. It’s a powerful demonstration of how aggregating many weak, unreliable signals can produce a single, highly reliable and actionable insight.

5.2 Confidence Intervals

The fact that the distribution of the simulated means from the Pyx example can be described well by a normal bell curve, in fact has a theoretical justification. It is called the *Central Limit Theorem*. The Central Limit Theorem states that, given a sufficiently large sample size, the distribution of the sample means will be approximately normally distributed, regardless of the shape of the population distribution. This normal distribution is also known as the Gaussian distribution.

There are a few conditions. The sampled observations must be independent. In practice, this means that the sampling should be random, and one observation should not influence another. Further, the sample size should be sufficiently large. While there is no strict rule for what constitutes ‘large enough,’ a common guideline is a sample size of 30 or more. However, if the population distribution is far from normal, a larger sample size may be required.

We can estimate the mean of this bell curve using \bar{x} and the standard deviation (standard error) using s/\sqrt{n} .

The square-root nature of this relation is somewhat unfortunate. To double your certainty about the population mean, you need to quadruple the sample size.

One of the main applications of this result is the construction of *confidence intervals*. A confidence interval is a range of values that is likely to contain the true value of the population mean. It is a plausible range for the quantity we are trying to estimate. The confidence interval is calculated using the sample mean \bar{x} and the standard error s/\sqrt{n} . The confidence interval is calculated as follows

$$\bar{x} \pm 1.96s_{\bar{x}}, s_{\bar{x}} = \frac{s}{\sqrt{n}}.$$

The theorem applies to a wide range of population distributions, including distributions that are not normal. This universality makes it one of the most powerful and widely-used theorems in statistics.

Here are a few conclusions we can make thus far

1. Mean estimates are based on random samples and therefore random (uncertain) themselves
We need to account for this uncertainty!
2. Standard Error measures the uncertainty of an estimate
3. Using properties of the Normal distribution, we can construct 95% Confidence Intervals

This provides us with a plausible range for the quantity we are trying to estimate.

Recall the Patriots coin toss example from Chapter 1, we know that they won 19 out of 25 tosses during the 2014-2015 season. In this example, our observations are values 0 (lost toss) and 1 (won toss) and the average over those 0-1 observations is called the *proportion* and is denoted by \hat{p} instead of \bar{x} . When we deal with proportions, we can calculate the sample variance from its mean \hat{p} as follows

$$s_{\hat{p}}^2 = \frac{\hat{p}(1 - \hat{p})}{n}.$$

Thus, we know that given our observations and CLT, the sampling distribution of our estimator \hat{p} is normal. Our best guess at the mean \hat{p} is $19/25 = 0.76$ and variance $s^2 = 0.76(1 - 0.76)/25 = 0.0073$

$$\hat{p} \sim N(0.76, 0.0073).$$

Then a 95% Confidence Interval is calculated by

```
## 0.59 0.93
```

Since 0.5 is outside the confidence interval, we say that we do not have enough evidence to say that the coin tosses were fair.

Then we formulate a hypothesis that we are to test. Our status-quo assumption (there is no effect) is called the null hypothesis and is typically denoted by H_0 .

5.2.1 Mythbusters Example

While A/B testing is industry-standard for software, the same methodology applies to testing physical phenomena or human behavior, dealing with small sample sizes where uncertainty is higher. In 2006 the creators of Mythbusters TV show on Discovery channel wanted to test whether yawning is contagious in humans. They recruited 50 participants and each of those went through an interview. At the end of 34 randomly selected interviews the interviewer did yawn. Then participants were asked to wait in a next door room. Out of 34 participants from the experimental group, 10 did yawn (29.4%) and only 4 out of 16 (25%) in the control group did yawn. The difference in the proportion of those who did yawn was 4.4%. The show hosts Kari Byron, Tory Belleci and Scottie Chapman concluded that yawn is indeed contagious.

To translate the question from this experiment into language of hypothesis testing, we say that our null hypothesis is that proportion of yawning participants in control (\hat{p}_c) and experimental group (\hat{p}_e) is the same $H_0 : \hat{p}_c - \hat{p}_e = 0$,

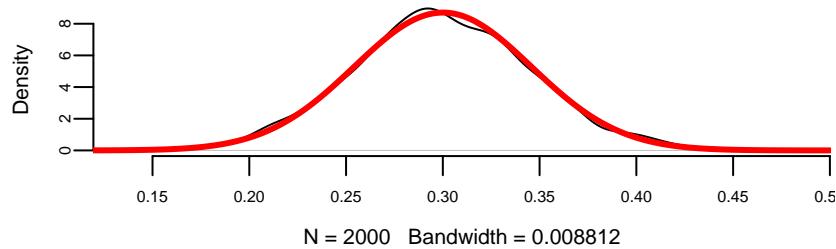
and the alternative hypothesis is $H_a : \hat{p}_c < \hat{p}_e$. The goal is to use the data to tell us if the hypothesis is correct or not.

A key statistical fact behind the hypothesis testing is the Central Limit Theorem. It states that if we have a sample $\{x_1, \dots, x_n\}$ with n observations from any distribution $x_i \sim p(x)$, then the average of the sample follows a Normal distribution with mean μ and variance σ^2/n

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i \sim N(\mu, \sigma^2/n)$$

Let us use a simple simulated data set to demonstrate the central limit theorem. We generate 100 outcomes of a Bernoulli trial with $p = 0.3$ and calculate the mean of this sample \hat{p} . We repeat it 2000 times and compare the empirical distribution of \hat{p} with $N(0.3, 0.046)$.

```
set.seed(1)
a <- replicate(2000, mean(rbinom(100, 1, 0.3)))
plot(density(a), main = "")
se <- sqrt(0.3 * (1 - 0.3) / 100) # 0.046
x <- seq(0, 0.5, length.out = 300)
lines(x, dnorm(x, mean = 0.3, sd = se), col = "red", lwd = 3)
```



There are three ways to quantify uncertainty in hypothesis testing. The first approach relies on calculating confidence intervals, as we did for the yawn example. There are two complementary approaches. One is to calculate what is called a *p*-value, that is the probability of getting the result observed in the data, assuming null-hypothesis is true. If *p*-value is low, then we reject the null-hypothesis. For the yawn example, the conditional probability that the observed difference in proportions is greater than 0.044, given null hypothesis is true is given by

$$p\text{-value} = P(\hat{p}_e - \hat{p}_c \geq 0.044 \mid H_0),$$

which can be calculated using `pnorm` function

```
1 - pnorm(0.044, 0, sqrt(0.0177))
## 0.37
```

The p -value of 0.37 means that there is a 37% chance to observe the difference to be greater than 0.044 assuming the null-hypothesis. It is quite high! We want the p -value to be low, only then we can claim that we have discovered a new fact, i.e. that yawning is contagious. In many applications we require this number to be at most 0.005. The smallest acceptable p -value is called the *significance level* and is typically denoted as α . We can test the hypothesis at different levels of significance α . Further we assume that the statistic we are analyzing follows the sampling distribution. The probability distribution of the statistics values is either Normal, or t -distribution for continuous variable.

Fundamentally, a statistical hypothesis is a testable statement about a population parameter that can be evaluated using observed data. To summarize the process of testing a significance of our discovery for proportions, we perform the hypothesis testing following the 5-step process.

- *Step 1:* Formulate the Null Hypothesis (H_0), which we assume to be true unless there is sufficient evidence to the contrary. Then, alternative Hypothesis (H_1): test against the null, e.g. $H_0 : p_e - p_c = 0$, and $H_a : p_e - p_c > 0$. If there is evidence that H_0 is false, we accept H_1 .
- *Step 2:* Select the significance level α . While $\alpha = 0.05$ (the 5% level) is the most commonly used, $\alpha = 0.01$ (the 1% level) is prevalent in medical and quality assurance examples.
- *Step 3:* Compute the Test Statistic (Z or T)
- *Step 4:* Formulate the Decision Rule. For example, reject the Null hypothesis if $|Z| > 1.96$
- *Step 5:* Make a Decision, Compute the p -value. p -value is the smallest significance level at which a null hypothesis can be rejected. If $p\text{-value} < \alpha$, we have evidence that H_1 is true, we accept H_1 and claim we have a discovery. If $p\text{-value} \geq \alpha$, then we cannot reject the null-hypothesis.

In Steps 1-2 we formulate the hypothesis. In steps 3-5 we make a decision.

In the context of hypothesis testing, we come back to the type I and type II errors we already discussed. They can be used to describe two types of errors you can make when testing

Error Type	Description
Type I Error	Rejecting a true H_0
Type II Error	Not rejecting a false H_0

And the significance level is then

$$P(\text{reject } H_0 \mid H_0 \text{ true}) = P(\text{type I error}).$$

Hypothesis testing is often used in scientific reporting. For example, the discovery of Higgs Boson was announced as a result of hypothesis testing. Scientists used the five-sigma concept to test the Higgs-Boson hypothesis. This concept, however, is somewhat counter-intuitive. If the particle doesn't exist, one in 3.5 million is the chance an experiment just like the one announced would nevertheless come up with a result appearing to confirm it does exist. In other words, one in 3.5 million is the likelihood of finding a false positive—a fluke produced by random statistical fluctuation that seems as definitive as the findings released by two teams of researchers at the CERN laboratory in Geneva. So we can talk about the significance level as p -value to be one-in-3.5-million and then the Z -score is five.

The test statistic (T or Z) quantifies uncertainty between the null-hypothesis value and the observed one and is equal to the number of standard deviations they are apart from each other. This value is called the Z -score, and is calculated as

$$Z = \frac{\bar{x} - \mu_0}{s/\sqrt{n}},$$

where μ_0 is the mean assumed under null-hypothesis. The square root of the statistic's variance s/\sqrt{n} is called standard error and is denoted by $se(\bar{X})$.

Let's calculate the Z -score for the yawning example. Let $\mu_0 = 0$, $\hat{p} = \hat{p}_e - \hat{p}_c = 0.044$, $\text{Var}(\hat{p}) = \text{Var}(\hat{p}_e - \hat{p}_c) = 0.0177$, we get Z statistic to be 0.33. Thus, our observed difference is very close to 0.

To summarize the duality of confidence interval, p -value and Z -score, the following statements are equivalent:

Statement	Condition
0 is inside the 95% confidence interval	p -value is greater than 0.05
p -value is greater than 0.05	Z -statistic is less than 1.96
Z -statistic is less than 1.96	0 is inside the 95% confidence interval

Example 5.3 (Stock market crash 1987 (Z-score)). Prior to the October, 1987 crash, SP500 monthly returns were 1.2% with a risk/volatility of 4.3%. The question is how extreme was the 1987 crash of -21.76%?

$$X \sim N(1.2, 4.3^2)$$

This probability distribution can be standardized to yield

$$Z = \frac{X - \mu}{\sigma} = \frac{X - 1.2}{4.3} \sim N(0, 1).$$

Now, we calculate the observed Z , given the outcome of the crash event

$$Z = \frac{-0.2176 - 0.012}{0.043} = -5.27$$

That is a 5-sigma event in terms of the distribution of X . Meaning that -0.2176 is 5 standard deviations away from the mean. Under a normal model that is equivalent to $P(X < -0.2176) = 4.6593 \times 10^{-8}$.

5.2.2 Other Applications

Here we consider three examples of hypothesis testing in action: the famous “Coke vs Pepsi” challenge, the dispute over the drug Avonex, and safety statistics for Viagra.

Example 5.4 (Coke vs Pepsi). The most famous hypothesis test in history in whether people can decide the difference between Coke and Pepsi. We run a double blind experiment, neither the experimenter or subject know the allocation. Pepsi claimed that more than half of Diet Coke drinkers said they preferred to drink Diet Pepsi. That is our null hypothesis. The data comes from a random sample of 100 drinkers. We find that 56 favor Pepsi.

This is a hypothesis test about the proportion of drinkers who prefer Pepsi

$$H_0 : p = \frac{1}{2} \text{ and } H_1 : p > \frac{1}{2}$$

Let's estimate our statistics form data:

$$\hat{p} = X/n = 56/100 = 0.56$$

This is my best estimate of the true p . The standard error of my statistic

$$se(\hat{p}) = \sqrt{\hat{p}(1 - \hat{p})/n} = 0.0496.$$

The 95% is then

$$0.56 \pm 1.96(0.0496) = 0.56 \pm 0.098 = (0.463, 0.657)$$

$p = 0.5$ lies inside the confidence interval. Pepsi was lying!

The Z -score now with $s_{\hat{p}} = \sqrt{p_0(1-p_0)/n} = 0.05$

$$Z = \frac{\hat{p} - p_0}{s_{\hat{p}}} = \frac{0.56 - 0.5}{0.05} = 1.2 < 1.64$$

Let's take the usual $\alpha = 0.05$.

```
prop.test(56, 100, alternative = "greater", conf.level = 0.95)
  %>% fmt_prop_test()
```

est	stat	p	param	low	high	alt
0.56	1.2	0.14		1	0.47	1 greater

Don't reject H_0 for a one-sided test at 5% level (p-value: 0.14). We need a larger n to come to a more definitive conclusion. We might come to a different conclusion with a larger sample size. One of the downsides of hypothesis testing is that it generates a yes/no answer without having any uncertainty associated with it.

Example 5.5 (Avonex). Biogen made the following assertion: “Avonex delivers the highest rate of satisfaction: 95% among patients” In response to that statement, the U.S. Food and Drug Administration (FDA) on October 30th, 2002 informed the biotech company Biogen to stop publishing misleading promotions for its multiple sclerosis drug Avonex. To clarify the issue, FDA did run an experiment. The FDA found that in a random sample of 75 patients surveyed, only 60% said they were satisfied with Avonex. The question is: Who is right?

Let's use hypothesis testing to get an answer. Following our five-step process to set up a Hypothesis Test: 1. Formulate the Null Hypothesis: $H_0 : p = 0.95 = p_0$. The alternative hypothesis: $H_1 : p < 0.95$. A 1-sided alternative. 2. We'll use a small significance level, 1%. 3. The appropriate test statistic is $Z = -14$.

To complete our analysis, we need to determine the critical region and calculate the p -value for this test. For a one-sided test at the 1% significance level, the critical region is $Z < -2.32$. Since our observed test statistic of $Z = -14$ falls well within this rejection region, we reject the null hypothesis.

This procedure can be implemented in R in the `prop.test` function.

```
prop.test(45, 75, 0.95) %>% fmt_prop_test()
```

est	stat	p	param	low	high	alt
0.6	186	0		1	0.48	0.71

The p-value is less than 2.2×10^{-16} , which is essentially zero. This extremely small p -value provides overwhelming statistical evidence that the FDA is correct and Biogen's claim is false.

Example 5.6 (Pfizer). Pfizer introduced Viagra in early 1998. During 1998 of the 6 million Viagra users 77 died from coronary problems such as heart attacks. Pfizer claimed that this rate is no more than the general population. A clinical study found 11 out of 1,500,000 men who were not on Viagra died of coronary problems during the same length of time as the 77 Viagra users who died in 1998. The question is whether the drug is safe. Let's calculate the confidence interval. A 95% confidence interval for a difference in proportions $p_1 - p_2$ is

$$(\hat{p}_1 - \hat{p}_2) \pm 1.96 \sqrt{\frac{\hat{p}_1(1 - \hat{p}_1)}{n_1} + \frac{\hat{p}_2(1 - \hat{p}_2)}{n_2}}$$

We can do a confidence interval or a Z-score test. With Viagra, $\hat{p}_1 = 77/6000000 = 0.00001283$ and without Viagra $\hat{p}_2 = 11/1500000 = 0.00000733$. We need to test whether these are equal. With a 95% confidence interval for $(p_1 - p_2)$ you get an interval $(0.00000549, 0.0000055)$.

Since this confidence interval does not contain zero, we have evidence that the proportion of deaths from coronary problems is significantly higher among Viagra users compared to the general population. Despite the small proportions involved, the measurement is highly accurate due to the large sample sizes in both groups. For hypothesis testing, we would use a one-sided test with a significance level of $\alpha = 0.01$ to test whether the proportion of deaths is higher in the Viagra group. The difference in proportions can be analyzed as follows:

```
prop.test(x = c(11, 77), n = c(1500000, 6000000), alternative =
  "greater", conf.level = .95) %>%
  fmt_prop_test()
```

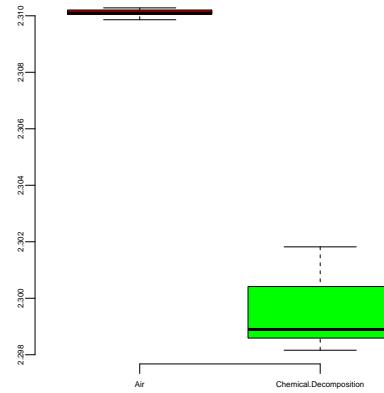
stat	p	param	low	high	alt
2.6	0.95		1	0	1 greater

The p-value for the Null is $1 - 0.948 = 0.052$. This p-value is greater than 0.01, so we cannot reject the null hypothesis.

Example 5.7 (Lord Rayleigh's Argon Discovery). Lord Rayleigh won the Nobel Prize for discovery of Argon. This discovery occurred when he noticed a small discrepancy between two sets of measurements on nitrogen gas that he had extracted from the air and one he had made in the lab.

First, he removed all oxygen from a sample of air. He measured the density of the remaining gas in a fixed volume at constant temperature and pressure. Second, he prepared the same volume of pure nitrogen by the chemical decomposition of nitrous oxide (N_2O) and nitric oxide NO . Here's the results. Table on the left shows the measurements. Boxplot on the right shows the distribution of the measurements.

Air	Chemical.Decomposition
2.3	2.3
2.3	2.3
2.3	2.3
2.3	2.3
2.3	2.3
2.3	2.3
2.3	2.3
NA	2.3



Although the averages are very close 2.31 vs 2.30, the small standard deviations (1.43×10^{-4} and 1.38×10^{-3}) suggest that the measurements are very precise.

```
t.test(air, decomp, var.equal = T)
##
##  Two Sample t-test
##
## data: air and decomp
## t = 20, df = 13, p-value = 0.000000000003
## alternative hypothesis: true difference in means is not equal
## to 0
## 95 percent confidence interval:
##  0.0095 0.0118
## sample estimates:
## mean of x mean of y
##      2.3      2.3
```

The t-test confirms that with the t-statistic of 20, the difference is statistically

significant. It is a 20-sigma event and we've found Argon!

5.3 A/B Testing Applications

While we have established the foundations of hypothesis testing, let's look at specific applications in industry.

Example 5.8 (SimCity). In the gaming industry, A/B testing is crucial for optimizing revenue. Electronic Arts is a company that makes video games. SimCity 5, one of EA's most popular video games, sold 1.1 million copies in the first two weeks of its launch. 50% of sales were digital downloads, thanks to a strong A/B testing strategy designed to maximize the conversion rate of pre-orders.

As EA prepared to release the new version of SimCity, they released a promotional offer to drive more game pre-orders. The offer was displayed as a banner across the top of the pre-order page – front-and-center for shoppers. But according to the team, the promotion was not driving the increase in pre-orders they had expected.

They decided to test some other options to see what design or layout would drive more revenue.



Figure 5.3: SimCity banner

One variation removed the promotional offer from the page altogether. The test led to some very surprising results: The variation with no offer messaging whatsoever drove 43.4% more purchases. Data revealed that customers

preferred a direct purchase path, indicating that additional incentives were unnecessary.

Most people believe that direct promotions drive purchases, but for EA, this turned out to be totally false. Testing gave them the information needed to maximize revenue in a way that would not have been otherwise possible.

We define the `abtestfunc` function to examine whether a black or pink background results in more purchases. Run experiment for one week:

- Pink background: 40% purchase rate with 500 visitors
- Black background: 30% purchase rate with 550 visitors

Let's run the AB test to see which is more effective, we will calculate the confidence intervals for conversion rates for each variation of site using the `abtestfunc` function.

The `abtestfunc` function below calculates CIs (80% significance, $Z = 1.28$).

```
## 37 43
## 28 32
```

The confidence intervals are [37, 42] for the pink background and [28, 32] for the black background. Since these intervals do not overlap, we can conclude that the purchase rate for the pink background is significantly higher than for the black background at the 80% confidence level.

Example 5.9 (Mythbusters). We now return to the Mythbusters yawning experiment to analyze it using the standard error of the difference in proportions.

The question is what happens if we are to re-run this experiment several times with different groups of participants, will we see the same difference of 4.4%? The fact is that from one experiment to another calculated proportions of yawners in both groups will be different.

In our example, the proportion of yawners in the experimental group is $\hat{p}_e = 0.294$ and in the control group is $\hat{p}_c = 0.25$. Thus,

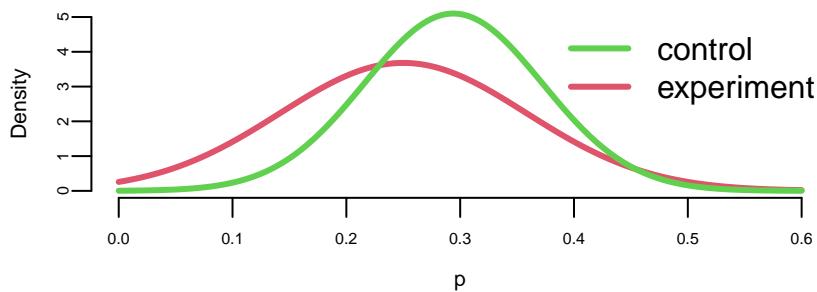
$$\hat{\sigma}_e^2 = 0.294(1 - 0.294) = 0.208, \quad \hat{\sigma}_c^2 = 0.25(1 - 0.25) = 0.188$$

We can apply CLT and calculate the uncertainty about \hat{p}_e and \hat{p}_c

$$\hat{p}_e \sim N(0.294, 0.208/34), \quad \hat{p}_c \sim N(0.25, 0.188/16).$$

Now, instead of comparing proportions (numbers), we can compare their distributions and thus quantify uncertainties. If we plot density functions of those two Normal variables, we can see that although means are different, there is a large overlap of the two density functions.

```
p <- seq(0.0, 0.6, length.out = 200)
plot(p, dnorm(p, 0.25, sqrt(0.188 / 16)), col = 2, type = "l",
  lwd = 3, ylab = "Density", ylim = c(0, 5))
lines(p, dnorm(p, 0.294, sqrt(0.208 / 34)), col = 3, lwd = 3,
  ylim = c(0, 6))
legend("topright", c("control", "experiment"), col = c(3, 2),
  lwd = 3, bty = "n")
```



The amount of overlap is the measure of how certain we are that p_e and p_c are different. Large overlap means we are not very certain if proportions are truly different. For example, both p_e and p_c have a high probability of being between 0.2 and 0.4. We can use properties of normal distribution to say specifically what is the amount of this overlap by calculating the corresponding 95% confidence interval of the difference between the proportions. Note that the difference of two Normal random variables is another Normal

$$\hat{p}_e - \hat{p}_c \sim N(0.294 - 0.25, 0.208/34 + 0.188/16) = N(0.044, 0.0177)$$

Now we can calculate 95% confidence interval for $\hat{p}_e - \hat{p}_c$, again using properties of Normal

```
## -0.22  0.30
```

The interval is wide and most importantly, it does contain 0. Thus, we cannot say for sure that the proportions are different. They might just appear to be different due to chance (sampling error). Meaning, that if we are to re-run the experiment we should expect the difference to be anywhere between -0.22 and 0.31 in 95% of the cases.

Thus, statistical analysis does not confirm the conclusion made by the show hosts and indicates that there is no evidence that the proportion of yawners is different between the control and experimental groups.

Example 5.10 (Search algorithm). For tech giants, even small improvements in metrics can lead to massive gains. Let's look at another example and test effectiveness of Google's new search algorithm. We measure effectiveness by the number of users who clicked on one of the search results. As users send the search requests, they will be randomly processed with Algo 1 or Algo 2. We wait until 2500 search requests were processed by each of the algorithms and calculate the following table based on how often people clicked through

Table 5.7: Google Search Algorithm

	Algo1	Algo2
success	1755	1828
failure	745	682
total	2500	2500

The probability of success is estimated to be $\hat{p}_1 = 1755/2500 = 0.702$ for the current algorithm and $\hat{p}_2 = 1828/2500 = 0.731$ for the new algorithm. We can calculate the 95% confidence interval or 95% Bayesian credible region for both estimated proportions.

```
p1 <- 1755 / 2500
p2 <- 1828 / 2500
Algo1 <- round(p1 + c(-1.96, 1.96) * sqrt(p1 * (1 - p1) / 2500),
  ↵ 3)
Algo2 <- round(p2 + c(-1.96, 1.96) * sqrt(p2 * (1 - p2) / 2500),
  ↵ 3)
kable(rbind(Algo1, Algo2), col.names = c("Lower", "Upper"),
  ↵ caption = "95% Confidence Interval for the Proportions")
```

Table 5.8: 95% Confidence Interval for the Proportions

	Lower	Upper
Algo1	0.68	0.72
Algo2	0.71	0.75

Given that the intervals do slightly overlap, there is not enough evidence to say that algorithms are different, and the new Algo 2 is not necessarily more efficient.

We will get a slightly more precise estimation of uncertainty if we calculate confidence interval for the difference of the proportions. Since p_1 and p_2 both follow Normal distribution, their difference is also normally distributed

$$p_1 - p_2 \sim N(\hat{p}_1 - \hat{p}_2, s_1^2/n + s_2^2/n).$$

Applying this formula for the Google search algorithm experiment, we calculate the 95% confidence interval for the difference

```
## -0.0541696 -0.0042304
```

The confidence interval for the difference does not contain 0, and thus we can say that we are confident that algorithms are different!

More generally, if the number of observations in two groups are different, say n_1 and n_2 then the

$$s_{\bar{X}_1 - \bar{X}_2} = \sqrt{\frac{s_{\bar{X}_1}^2}{n_1} + \frac{s_{\bar{X}_2}^2}{n_2}}$$

or for proportions, we compute

$$s_{\hat{p}_1 - \hat{p}_2} = \sqrt{\frac{\hat{p}_1(1 - \hat{p}_1)}{n_1} + \frac{\hat{p}_2(1 - \hat{p}_2)}{n_2}}.$$

Example 5.11 (Search Algorithm with Unequal Sample Sizes). Let's consider a variation of the previous example where the sample sizes are different. Suppose we have the following data:

Table 5.9: Google Search Algorithm with Unequal Sample Sizes

	Algo1	Algo2
success	1824	1867
failure	776	683
total	2600	2550

The probability of success is estimated to be $\hat{p}_1 = 1824/2600 = 0.702$ for Algo1 and $\hat{p}_2 = 1867/2550 = 0.732$ for Algo2. Now we calculate the 95% confidence interval for the difference in proportions using the formula for unequal sample sizes:

```
p1 <- 1824 / 2600
p2 <- 1867 / 2550
diff <- p1 - p2 + c(-1.96, 1.96) * sqrt(p1 * (1 - p1) / 2600 +
  ↵ p2 * (1 - p2) / 2550)
print(diff, digits = 5)
## -0.0552111 -0.0060257
```

The confidence interval for the difference does not contain 0, and thus we can say that we are confident that algorithms are different!

5.4 Challenges in A/B Testing

5.4.1 Multiple Testing

Consider this simple multiple-testing scenario. If we want to test 1000 hypotheses and we test each hypothesis one-by-one, say the ground truth is that only 10% (100) of those hypotheses are true. Using $\alpha = 0.05$ rule, we assume that out of 900 false hypotheses $0.05 \cdot 900 = 45$ will show up as positive (false positives). Now we run our one-by-one hypothesis tests and our procedure correctly identified 80 out of 100 true positives and incorrectly identified 45 false positives and 20 false negatives. Now, among 125 hypotheses identified as positives 45 in fact are not! Another way to look at it is to calculate the probability of at least one false positive $P(\text{at least one false positive}) = 1 - (1 - 0.05)^{1000} = 1$. We are almost guaranteed to see at least one false positive.

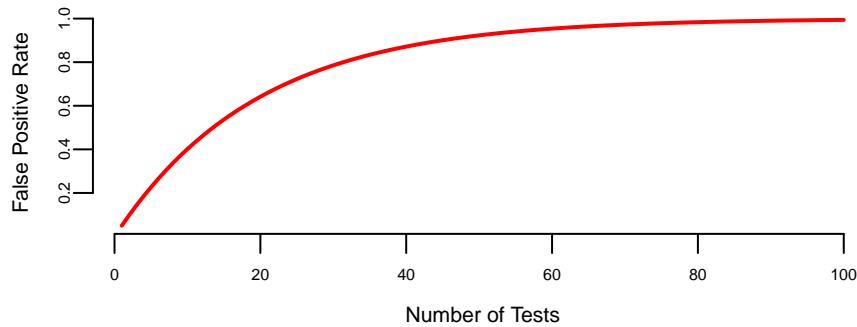


Figure 5.4: Probability of At Least 1 False Positive

One way to deal with the problem is to lower the cut-off to α/n . This approach is called the Bonferroni correction. For the case of 1000 hypotheses we set $\alpha = 0.00005$. However this conservative approach will lead to many false negatives. The probability of identifying at least one significant result is then $1 - (1 - 0.00005)^{1000} = 0.049$

Table 5.10: Classification of results for a testing procedure. T/F = True/False, D/N = Discovery/Non-discovery. We observe m , D and N .

	H_0 Accepted	H_0 Rejected	Total
H_0 True	TN	FD	T_0
H_0 False	FN	TD	T_1
Total	N	D	m

	H_0 Accepted	H_0 Rejected	Total
--	----------------	----------------	-------

A more practical approach is to use the *False Discovery Rate*

$$\text{FDR} = \text{E} \left(\frac{FD}{D} \right)$$

which is the proportion of false positives among all significant results. We aim to set a cutoff so that $\text{FDR} < Q$. The FDR approach allows us to increase the power while maintaining some principled bound on error.

Benjamini and Hochberg developed a procedure based on FDR to perform multiple testing. Under their procedure, we put individual p -values in order from smallest to largest. Then we choose the largest p_k value that is smaller than $(k/m)/Q$ where Q is the false discovery rate you choose. Then all hypotheses with index $i < k$ are significant. Benjamini and Hochberg showed that under this procedure the $\text{FDR} < Q$.

As an example, García-Arenzana et al. (2014) tested associations of 25 dietary variables with mammographic density, an important risk factor for breast cancer, in Spanish women. They found the following results:

Table 5.11: Dietary Risk Factors of Cancer

Label	p.value	Rank	BH
Total calories	0.00	1	0.01
Olive oil	0.01	2	0.02
Whole milk	0.04	3	0.03
White meat	0.04	4	0.04
Proteins	0.04	5	0.05

If we choose $Q = 0.25$, then $k = 5$ (Proteins) is our cut-off rank. Thus we reject H_0 for the first five tests. Note that traditional hypothesis testing procedure only controls for Type 1 error and FDR-based procedure controls for both error types.

5.5 Randomized Controlled Trials and Causality

Florence Nightingale (1820-1910), widely known for her role in the Crimean War, was also a pioneering statistician and a champion of evidence-based

medicine. She can be considered the mother of observational studies. To her, every piece of legislation was an experiment in the laboratory of society deserving study and demanding evaluation. Nightingale recognized the importance of collecting accurate and reliable data to understand healthcare outcomes. She developed standardized methods for collecting data on hospital admissions, deaths, causes of death, and other relevant factors. This systematic data collection allowed for more rigorous and reliable analysis of healthcare practices and their impact on patient outcomes. During the Crimean War (1853-1856), she collected and analyzed data on mortality rates among soldiers. She created statistical diagrams, such as the famous polar area diagram or “coxcomb,” to illustrate the causes of mortality. These visual representations helped to convey complex information in a clear and understandable way. Nightingale’s observations and statistical analyses led her to emphasize the importance of sanitation and hygiene in healthcare settings. She advocated for improvements in cleanliness, ventilation, and sanitation in hospitals, recognizing the impact of these factors on the health and well-being of patients. Beyond the battlefield, Nightingale continued her work in public health. She used statistical evidence to advocate for healthcare reforms and improvements in public health infrastructure. Her efforts played a crucial role in shaping public health policies and practices.

The work of Nightingale would nowadays be classified as an observational study. An observational study is a research design where researchers observe and collect data on existing groups of people or phenomena without intervening or manipulating any variables. Unlike randomized controlled trials, researchers do not assign participants to different groups and do not directly influence the outcome.

In contrast, George Washington (1732-1799) advanced agriculture through rigorous experimentation, employing methods remarkably similar to modern controlled experiments. He was deeply interested in improving agricultural techniques and conducted numerous experiments at his Mount Vernon estate. One of his most notable experiments involved dividing his land into plots and testing different crop rotations and fertilization methods. Washington recognized the importance of sustainable agriculture and the detrimental effects of monoculture (growing the same crop year after year) on soil fertility. He observed how tobacco, his primary cash crop at the time, depleted the soil nutrients, leading to diminishing yields. To address this issue and improve the long-term health of his land, he began experimenting with crop rotation and soil management techniques.

Washington divided his land into several plots, each receiving different treatments. He experimented with various crop rotations, including wheat-fallow, wheat-rye-fallow, and corn-wheat-fallow. These rotations aimed to prevent soil depletion and promote its natural restoration by planting nitrogen-fixing crops like rye and clover. He also tested different fertilizer applications on various plots. He used manure, compost, and even imported materials like gypsum

and marl to improve soil fertility and crop yields.

Washington meticulously documented his experiments in his agricultural diaries. He recorded planting dates, yields, weather conditions, and observations on crop growth and soil health. This meticulous record-keeping allowed him to analyze the effectiveness of different treatments and compare their impact on crop yields and soil quality.

Washington's experiments yielded valuable insights into sustainable agricultural practices. He discovered that crop rotation and fertilization improved soil health and increased crop yields over time. He abandoned tobacco as his primary crop and shifted towards wheat, which was less soil-depleting and offered a more stable income source.

The historic trades staff at Mount Vernon have recreated Washington's experiment at the Pioneer Farm, using the same plot layout, crops, and fertilization methods described in his diaries. This allows visitors to learn about his innovative farming techniques and their impact on the land. Figure 5.5 shows the plot layout at the Pioneer Farm.



Figure 5.5: Plot layout at the Mount Vernon's Pioneer Farm

George Washington's commitment to experimentation and innovation made him a pioneer in American agriculture. His plot-based experiments demonstrated the effectiveness of crop rotation and soil management in promoting sustainable farming practices. His work continues to inspire farmers today and serves as a valuable resource for understanding agricultural history and best practices.

Later, at the turn of the 20th century, Ronald Fisher (1890-1962) developed the theory of experimental design which allowed for controlled experiments, known as randomized controlled trials (RCT). Fisher's work laid the foundation for modern experimental design and analysis, providing a rigorous statistical framework for conducting randomized controlled trials. His contributions to experimental design and ANOVA were crucial in establishing the importance of randomized trials in research. He emphasized the importance of randomization and control groups in experimental design, recognizing their crucial role in establishing causal relationships.

5.5.1 The Question of Causation

Randomized controlled trials (RCTs) and field experiments are widely considered the gold standard for establishing causation because they allow researchers to isolate the effect of a specific intervention or treatment from other confounding factors. The importance of distinguishing causation from correlation is not merely academic—it lies at the heart of rational decision-making. As Newcomb's problem illustrates (Example 4.2), even the definition of “rational choice” becomes contentious when the causal structure between decisions and outcomes is unclear. In that paradox, whether you should “one-box” or “two-box” depends entirely on whether your choice *causes* the predictor’s action or merely *correlates* with it. RCTs resolve such ambiguities by ensuring that treatment assignment is causally independent of potential outcomes. The main principle of RCTs and field experiments is randomization, which ensures that the treatment and control groups are similar in all respects except for the treatment. This allows researchers to attribute any differences in outcomes between the two groups to the treatment, rather than to other factors.

Randomization helps to control for confounding variables, which are factors that are associated with both the treatment and the outcome variable. By randomly assigning participants to groups, researchers can ensure that any confounding variables are evenly distributed between the groups. The control group serves as a baseline for comparison. It is a group that is not exposed to the treatment or intervention being studied. By comparing the outcomes of the treatment group and the control group, researchers can isolate the effect of the treatment. Any differences in the outcomes between the two groups can be attributed to the treatment.

From a Bayesian modeling perspective, randomization is a practical route to approximate exchangeability between groups, making comparisons less sensitive to unobserved confounding (Chapter 3).

The modern randomized controlled trial (RCT) in medicine is most often attributed to Sir Austin Bradford Hill. In 1948, Hill published a landmark paper titled “Streptomycin Treatment of Pulmonary Tuberculosis” in the British Medical Journal, which described the first fully randomized, double-blind clinical trial. This study is considered a turning point in the history of medical research and established the RCT as the gold standard for evaluating the effectiveness of medical treatments.

Randomized trials and observational studies are two distinct approaches to gathering and analyzing data in research studies. Here’s a breakdown of their key differences:

If you happen to have a choice between randomized trials and observational data (often you do not have that choice), which one should you choose? Consider the following:

Randomized trials

- *Definition:* Participants are randomly assigned to different groups, with one group receiving the intervention being studied and the other group receiving a control intervention or placebo.
- *Purpose:* Determine whether the intervention causes the observed outcome by controlling for other factors that might influence the results.
- *Strengths:* High internal validity, strong causal inference due to randomization, allows for isolating the effect of the intervention, minimizes selection bias through random assignment.
- *Weaknesses:* Can be expensive and time-consuming to conduct, may not be ethical or feasible for all interventions, may not be generalizable to real-world settings.
- *Research question:* If the research question aims to establish causation, a randomized trial is generally preferred. However, if the goal is to explore associations or generate hypotheses, observational data may be sufficient.
- *Available resources:* Randomized trials require significant resources, while observational studies can be less expensive and time-consuming.
- *Ethical considerations:* Randomizing individuals to certain interventions may be unethical, making observational data the only option.
- *Generalizability:* Randomized trials often involve carefully controlled environments, which may limit their generalizability to real-world settings. Observational data can provide insights into how interventions work in real-world situations.

Observational data

- *Definition:* Data is collected on existing groups of people without any intervention being implemented. Researchers observe and analyze the data to identify relationships between variables.
- *Purpose:* Explore potential associations between variables, generate hypotheses for further research, and investigate the natural course of a phenomenon.
- *Strengths:* Often less expensive and time-consuming, can provide insights into real-world settings, can investigate interventions that are not ethically feasible to test in randomized trials.
- *Weaknesses:* Lower internal validity, susceptibility to confounding variables, cannot establish causal relationships conclusively.

Ultimately, both randomized trials and observational data play crucial roles in research. Combining these two approaches provides a more comprehensive understanding of the relationship between interventions and outcomes.

Example 5.12 (Russian election fraud: a field experiment). Enikolopov et al. (2013) show how a field experiment can be used to estimate electoral fraud in Russian parliamentary elections held on December 4, 2011. They randomly assigned independent observers to 156 of 3,164 polling stations in the city of Moscow. The observers were trained by the nongovernmental organization Citizen Observer. The authors compared the vote shares of the incumbent

United Russia party at polling stations with and without observers. They found that the presence of observers decreased the reported vote share of United Russia by almost 11 percentage points. This suggests that the extent of the fraud was sufficient to have changed the outcome of the elections.

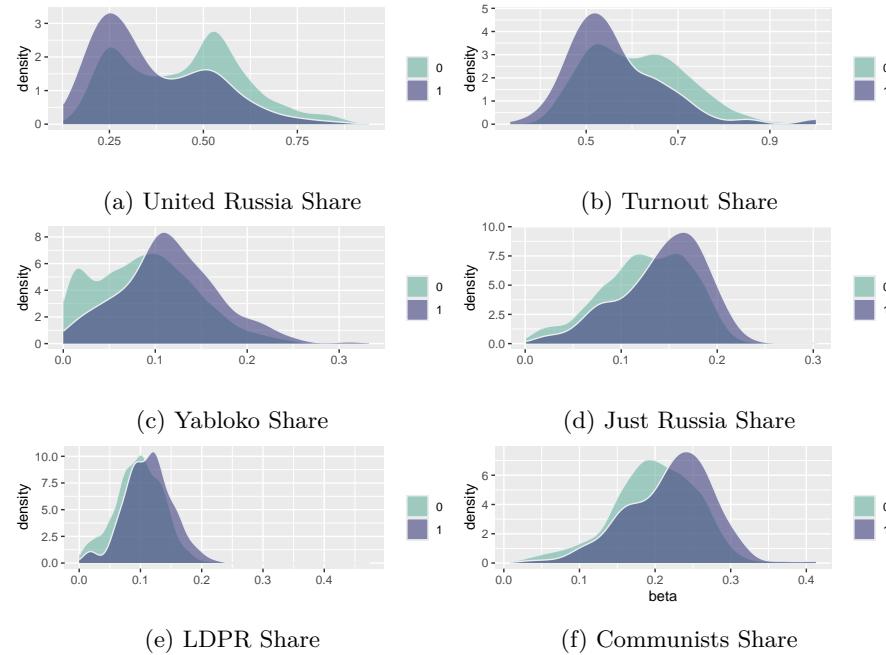


Figure 5.6: Histogram comparison of the share of votes received by different parties and the share of those eligible voters who actually voted (turnout)

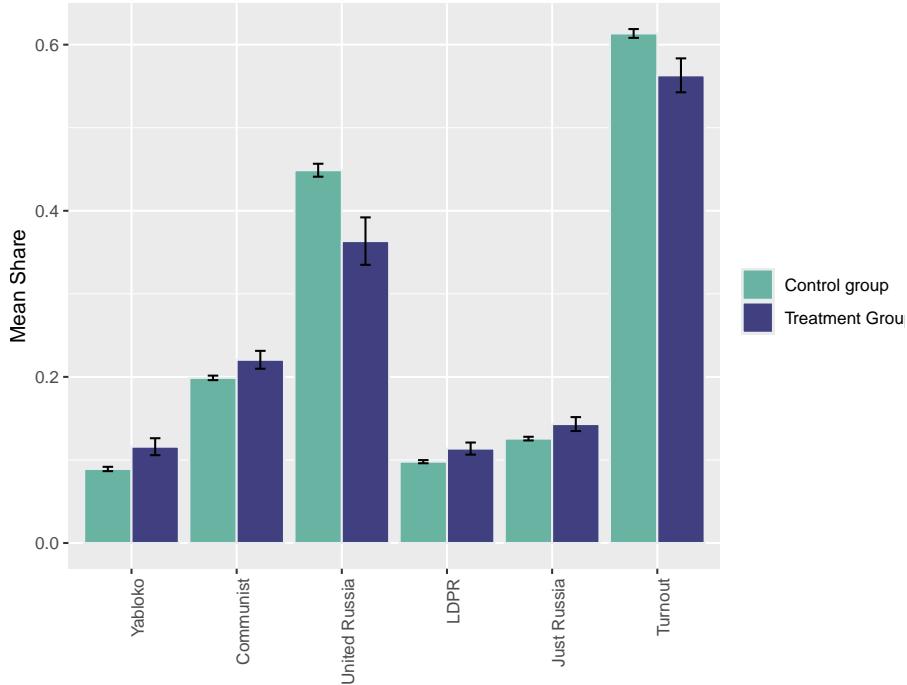


Figure 5.7: Bar plot comparison of the share of votes received by different parties and the share of those eligible voters who actually voted (turnout)

Figure 5.6 and Figure 5.7 show the results of the experiments and plot histograms of the vote shares. The first histogram compares the share of the ruling United Russia party at the polling stations without observers ($treatment = 0$) and with observers ($treatment = 1$). On average, United Russia vote share is decreased by 11 percent when observers were present. The calculations made by Enikolopov et al. (2013) showed that this amount of manipulation was enough to preserve the majority of United Russia in the parliament; it would have lost it without manipulations. While “adding” votes for UR, the results indicate that all other parties were hurt by electoral fraud. The Liberal Democratic Party of Russia (LDPR) was hurt the least and is believed to be the most loyal to the ruling party.

Example 5.13 (Pollution in India). Randomized Controlled Trials (RCTs) have transformed economic research and policy-making by providing a rigorous methodology to establish causal relationships between interventions and outcomes. The 2019 Nobel Prize in Economics awarded to Esther Duflo, Abhijit Banerjee, and Michael Kremer recognized their pioneering work in applying experimental approaches to alleviating global poverty and transforming development economics. Their experimental approach has fundamentally changed how economists tackle complex social problems by breaking them down into

smaller, more manageable questions that can be answered through carefully designed experiments.

The paper Duflo et al. (2013) exemplifies this experimental approach. This two-year field experiment conducted in Gujarat, India, examined how altering the market structure for environmental audits could improve the accuracy of pollution reporting and ultimately reduce industrial pollution. The study demonstrates how RCTs can identify causal mechanisms in complex regulatory environments and provide evidence for effective policy reforms.

The researchers randomly assigned 473 industrial plants to either a treatment or control group. In the treatment group, they implemented a package of reforms to the environmental audit system:

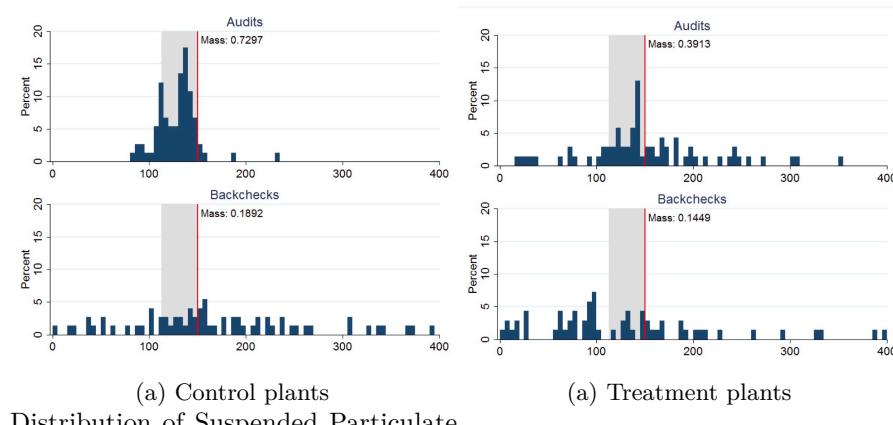
Reform	Description
<i>Random Assignment</i>	Auditors were randomly assigned to plants (rather than plants choosing their auditors)
<i>Central Payment</i>	Auditors were paid from a central pool at a fixed rate
<i>Random Backchecks</i>	Independent technical agencies conducted random backchecks of auditor reports
<i>Incentive Pay</i>	In the second year, incentive pay was provided for accurate reporting

In the control group, auditors were paid by the plants they audited. This created a blatant conflict of interest: plants could simply hire auditors who would provide favorable reports, leading to systematic misreporting of pollution levels. In contrast, the treatment group broke this client-auditor relationship by paying auditors from a central pool and independently verifying their work. The random backchecks served as a robust deterrent against misreporting, while the incentive pay further encouraged honesty.

Further, researchers performed backchecks, which are independent verification. Backchecks are follow-up visits conducted by independent technical agencies to verify the accuracy of pollution readings reported by third-party auditors. They serve as a quality control mechanism to monitor whether auditors are truthfully reporting actual pollution levels or manipulating data to show false compliance with regulatory standards.

The figure below (copied from the original paper) displays the distribution of Suspended Particulate Matter (SPM) concentrations measured in boiler-stack samples during the midline survey. Left plot presents the distributions of readings from both audits and backchecks at control plants, while right plot presents the corresponding distributions for treatment plants. A vertical line indicates the regulatory maximum concentration limit of 150 mg/N m³ for

SPM, with the region between 75% and 100% of this limit highlighted in gray shading.



Distribution of Suspended Particulate Matter. Source: Duflo et al. (2013).

This figure clearly demonstrates how the RCT revealed systematic misreporting in the status quo audit system and how the treatment intervention improved reporting accuracy. The stark difference between audit reports and backcheck readings in the control group (Panel A) provides visual evidence of corruption that would have been difficult to establish through observational methods alone.

The experiment yielded three main results. First, regarding status quo corruption, under the existing system, auditors systematically reported plant emissions just below the regulatory standard, even though true emissions were typically higher. Second, the treatment improved reporting accuracy, causing auditors to report pollution levels more truthfully, with treatment auditors reporting pollution readings 50-70% higher than control auditors. Third, plants in the treatment group reduced their actual emissions by 0.21 standard deviations, with reductions concentrated among the highest-polluting plants.

The work of Duflo, Banerjee, Kremer, and their collaborators has fundamentally changed how economists approach questions of causality and policy effectiveness. By adapting experimental methods from medical research to address economic and social questions, they have created a powerful toolkit for identifying effective interventions to address poverty and other global challenges.

The Gujarat environmental audit experiment exemplifies how RCTs can uncover hidden mechanisms—in this case, corruption in regulatory reporting—and test solutions that might not have been evident from observational data alone. The study's findings demonstrate that reformed incentives for third-party auditors can improve their reporting and make regulation more effective,

with tangible benefits for environmental quality.

As RCTs continue to evolve and spread across different domains of economics, they promise to further strengthen the evidence base for policy decisions, ultimately leading to more effective interventions and better outcomes for society.

5.5.2 Split-plot designs and Rothamsted

While Fisher championed randomization, practical constraints in agriculture often made complete randomization difficult. At the Rothamsted Experimental Station, where Fisher worked, researchers frequently encountered situations where some factors were difficult or expensive to randomize across small individual plots, while others were easy.

For example, consider an experiment testing the effects of two factors: irrigation methods (Factor A) and fertilizer types (Factor B). Applying different irrigation methods (e.g., flood vs. drip) typically requires large equipment and infrastructure, making it impractical to switch methods for every small plot of land. In contrast, changing fertilizer types is relatively easy and can be done by hand on smaller patches.

If we were to use a completely randomized design, we would have to randomly assign irrigation and fertilizer combinations to every small plot, which might require an impossible amount of plumbing/setup changes. Fisher and his colleagues formalized the split-plot design to handle this.

In a split-plot design, the experimental units are nested: 1. Whole Plots: Large areas of land to which the “hard-to-change” factor (e.g., irrigation) is applied. 2. Subplots: Smaller divisions within each whole plot to which the “easy-to-change” factor (e.g., fertilizer) is applied.

This design is a compromise. By restricting randomization of the main factor to larger blocks, we sacrifice some precision in estimating the main effect of Factor A (irrigation). However, we gain higher precision in estimating the effect of Factor B (fertilizer) and the interaction between A and B, because comparisons between fertilizers are made within the same whole plot, holding the irrigation method constant.

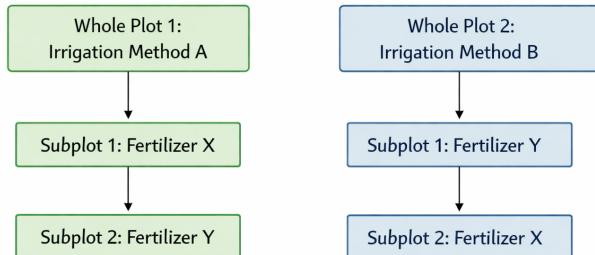


Figure 5.10: Split-plot design

This hierarchical structure is not limited to agriculture. In modern industrial and medical experiments, we often see similar structures. For instance, in a hospital trial, a new sanitation protocol (Factor A) might be applied to entire hospital wings (Whole Plots) because it's hard to have different protocols for adjacent rooms, while individual patients within those wings (Subplots) might be randomized to receive different medications (Factor B). Understanding these constraints is crucial ensuring that the statistical analysis accounts for the correlations among subplots within the same whole plot; ignoring this nesting leads to underestimated standard errors and overconfident conclusions.

5.5.3 Digital experimentation (A/B testing)

For many years, the main areas of application of randomized trials were medicine and agriculture. However, with the rise of the internet, randomized trials have become increasingly popular for testing the effectiveness of online interventions, such as email campaigns, website designs, and social media ads. When applied to user experience and marketing, randomized trials are often called A/B tests. The idea of A/B testing is the same as in traditional RCTs: randomly assign users to different versions of a website or an email campaign and compare the outcomes.

However, digital A/B tests often operate with fewer constraints on randomization protocols and sample selection than clinical or agricultural trials. There are fewer strict rules about ethics, randomization, sample size, and statistical analysis. For example, randomization is sometimes completely ignored in A/B testing. Instead of assigning users randomly to groups, they are divided into groups based on factors like time of day, location, or browsing history. This can introduce bias into the results, as the groups may not be comparable. As a result, A/B testing is cheap and quick to conduct, as it can be done online without the need for IRB approval or recruitment of participants. A/B testing is primarily focused on measuring the comparative performance of variations without necessarily establishing a causal relationship (e.g., “Which version of our web page leads to more clicks?”).

5.5.4 The era of observational data

Yet, not all digital data comes from structured experiments. We rely heavily on observational data streams. Diane Lambert was a pioneer in promoting proper statistical methods for observational data in the tech industry. She highlighted how to detect selection bias in data streams drawn from transaction logs and ad systems, and how to use diagnostics to judge when confounding bias is severe enough to overwhelm the causal signal. She advocated for practical fixes such as propensity-score weighting and simulation to ensure that credible causal conclusions can still be drawn from field data.

The advent of digital data has fundamentally transformed the practice of statistics, shifting the field from a discipline focused on small, carefully collected samples to one that must grapple with massive, often messy datasets generated as byproducts of digital systems. In the pre-digital era, statisticians worked primarily with structured, purposefully collected data through surveys, experiments, and clinical trials. Today, organizations routinely collect terabytes of data from web traffic, sensor networks, financial transactions, and social media interactions. This shift has necessitated new statistical approaches that can handle high-dimensional data and complex dependencies. Machine learning algorithms have become essential tools for extracting patterns from these vast datasets. However, this transition has also introduced new challenges: the need to distinguish correlation from causation in observational data, the importance of addressing selection bias in non-random samples, and the ethical considerations of privacy and algorithmic fairness.

5.5.5 From experiments to observational data

While randomized controlled trials remain the gold standard for establishing causality, conducting them is not always feasible. In many real-world scenarios—especially in policy, economics, and social science—we cannot ethically or practically assign treatments randomly. We are often left with observational data, where the “treatment” (e.g., a job training program, a medical procedure, or a new law) was assigned based on complex, often unobserved, human decisions. This brings us back to the fundamental problem of causal inference: how do we estimate the effect of a treatment when the treated and control groups are systematically different? To solve this, we need rigorous statistical frameworks that can adjust for these differences and impute the missing counterfactual outcomes.

Tree-based Bayesian methods are one modern approach to this problem. In particular, BART-based causal inference is developed in the tree chapter; see Section 14.5.



6

Bayesian Hypothesis Testing

The hypothesis testing problem is as follows. Based on a sample of data, y , generated from $p(y | \theta)$ for $\theta \in \Theta$, the goal is to determine if θ lies in Θ_0 or in Θ_1 , two disjoint subsets of Θ . In general, the hypothesis testing problem involves an action: accepting or rejecting a hypothesis. The problem is described in terms of a null, H_0 , and alternative hypothesis, H_1 , which are defined as

$$H_0 : \theta \in \Theta_0 \text{ and } H_1 : \theta \in \Theta_1.$$

As a scope note, we will be explicit about notation: in this chapter α and β refer to Type I and Type II error probabilities in the classical sense, defined through repeated-sampling performance of a decision rule. When we discuss Bayesian evidence, we instead emphasize posterior probabilities and Bayes factors, and when we talk about posterior uncertainty we use credible intervals and posterior summaries without reusing α as a credibility-level parameter.

Different types of regions generate different types of hypothesis tests. If the null hypothesis assumes that Θ_0 is a single point, $\Theta_0 = \{\theta_0\}$, this is known as a simple or “sharp” null hypothesis. If the region consists of multiple points, the hypothesis is called composite; this occurs when the space is unconstrained or corresponds to an interval of the real line. In the case of a single parameter, typical one-sided tests are of the form $H_0 : \theta < \theta_0$ and $H_1 : \theta > \theta_0$.

There are two correct decisions and two possible types of errors. The correct decisions are accepting a null or an alternative that is true, whereas a Type I error incorrectly rejects a true null and a Type II error incorrectly accepts a false null.

	$\theta \in \Theta_0$	$\theta \in \Theta_1$
Accept H_0	Correct decision	Type II error
Accept H_1	Type I error	Correct decision

Formally, the probabilities of Type I (α) and Type II (β) errors are defined as:

$$\alpha = P[\text{reject } H_0 \mid H_0 \text{ is true}] \text{ and } \beta = P[\text{accept } H_0 \mid H_1 \text{ is true}].$$

It is useful to think of the decision to accept or reject as a decision rule, $d(y)$. In many cases, the decision rules form a critical region R , such that $d(y) = d_1$ if $y \in R$. These regions often take the form of simple inequalities. Next, defining the decision to accept the null as $d(y) = d_0$, and the decision to accept the alternative as d_1 , the error types are

$$\begin{aligned}\alpha_\theta(d) &= P[d(y) = d_1 | \theta] \text{ if } \theta \in \Theta_0 \text{ (H_0 is true)} \\ \beta_\theta(d) &= P[d(y) = d_0 | \theta] \text{ if } \theta \in \Theta_1 \text{ (H_1 is true).}\end{aligned}$$

where both types of errors explicitly depend on the decision and the true parameter value. Notice that both of these quantities are determined by the population properties of the data. In the case of a composite null hypothesis, the size of the test (the probability of making a type I error) is defined as

$$\alpha = \sup_{\theta \in \Theta_0} \alpha_\theta(d)$$

The supremum (sup) is the least upper bound of a set. For finite sets, sup = max. For a standard reference, see Billingsley (1995). and the power is defined as $1 - \beta_\theta(d)$. It is always possible to set either $\alpha_\theta(d)$ or $\beta_\theta(d)$ equal to zero, by finding a test that always rejects the alternative or null, respectively.

The total probability of making an error is $\alpha_\theta(d) + \beta_\theta(d)$, and ideally one would seek to minimize the total error probability, absent additional information. The optimal action d^* minimizes the posterior expected loss; $d^* = d_0 = 0$ if the posterior probability of hypothesis H_0 exceeds 1/2, and $d^* = d_1 = 1$ otherwise

$$d^* = 1(P(\theta \in \Theta_0 | y) < P(\theta \in \Theta_1 | y)) = 1(P(\theta \in \Theta_0 | y) < 1/2).$$

Formally, a decision rule selects the hypothesis with higher posterior probability.

The easiest way to reduce the error probability is to gather more data, as the additional evidence should lead to more accurate decisions. In some cases, it is easy to characterize optimal tests, those that minimize the sum of the errors. Simple hypothesis tests of the form $H_0 : \theta = \theta_0$ versus $H_1 : \theta = \theta_1$, are one such case admitting optimal tests. Defining d^* as a test accepting H_0 if $a_0 f(y | \theta_0) > a_1 f(y | \theta_1)$ and H_1 if $a_0 f(y | \theta_0) < a_1 f(y | \theta_1)$, for some a_0 and a_1 . Either H_0 or H_1 can be accepted if $a_0 f(y | \theta_0) = a_1 f(y | \theta_1)$. Then, for any other test d , it is not hard to show that

$$a_0 \alpha(d^*) + a_1 \beta(d^*) \leq a_0 \alpha(d) + a_1 \beta(d),$$

where $\alpha_d = \alpha_d(\theta)$ and $\beta_d = \beta_d(\theta)$. This result highlights the optimality of tests defining rejection regions in terms of the likelihood ratio statistic, $f(y | \theta_0)/f(y | \theta_1)$. It turns out that the results are in fact stronger. In terms of decision theoretic properties, tests that define rejection regions based on

likelihood ratios are not only admissible decisions, but form a minimal complete class, the strongest property possible.

One of the main problems in hypothesis testing is that there is often a tradeoff between the two goals of reducing type I and type II errors: decreasing α leads to an increase in β , and vice-versa. Because of this, it is common to fix $\alpha_\theta(d)$, or $\sup \alpha_\theta(d)$, and then find a test to minimize $\beta_d(\theta)$. This leads to “most powerful” tests. There is an important result from decision theory: test procedures that use the same size level of α in problems with different sample sizes are inadmissible. This is commonly done where significance is indicated by a fixed size, say 5%. The implications of this will be clearer below in examples.

6.1 Likelihood Principle

Given observed data y and likelihood function $l(\theta) = p(y | \theta)$, the likelihood principle states that all relevant experimental information is contained in the likelihood function for the observed y . Furthermore, two likelihood functions contain the same information about θ if they are proportional to each other. For example, the widely used maximum-likelihood estimation does satisfy the likelihood principle. However, frequentist hypothesis testing procedures often violate this principle. The likelihood principle is a fundamental principle in statistical inference, and it is a key reason why Bayesian procedures are often preferred.

Decision Theoretic Concepts

- **Admissibility:** A decision rule δ is *admissible* if there exists no other rule δ' such that $R(\theta, \delta') \leq R(\theta, \delta)$ for all θ , with strict inequality for at least one θ . In other words, an admissible rule cannot be uniformly improved upon.
- **Complete Class:** A class of rules \mathcal{C} is *essentially complete* if for any rule $\delta \notin \mathcal{C}$, there exists a rule $\delta' \in \mathcal{C}$ that dominates it. A *minimal complete class* is the smallest such set of rules containing all admissible rules.

Example 6.1 (Testing fairness). Suppose we are interested in testing θ , the unknown probability of heads for a possibly biased coin. Suppose,

$$H_0 : \theta = 1/2 \quad \text{v.s.} \quad H_1 : \theta > 1/2.$$

An experiment is conducted and 9 heads and 3 tails are observed. This in-

formation is not sufficient to fully specify the model $p(y | \theta)$. There are two approaches.

Scenario 1: Number of flips, $n = 12$ is predetermined. Then number of heads $Y | \theta$ is binomial $B(n, \theta)$, with probability mass function

$$p(y | \theta) = \binom{n}{y} \theta^y (1 - \theta)^{n-y} = 220 \cdot \theta^9 (1 - \theta)^3$$

For a frequentist, the p-value of the test is

$$P(Y \geq 9 | H_0) = \sum_{y=9}^{12} \binom{12}{y} (1/2)^y (1-1/2)^{12-y} = (1+12+66+220)/2^{12} = 0.073,$$

and if you recall the classical testing, H_0 is not rejected at level $\alpha = 0.05$.

Scenario 2: The number of tails (successes) $r = 3$ is predetermined; that is, flipping continues until 3 tails are observed. Then, Y , the number of heads (failures) observed until 3 tails appear, follows a Negative Binomial distribution $NB(3, 1 - \theta)$,

$$p(y | \theta) = \binom{r+y-1}{r-1} \theta^y (1 - \theta)^r = \binom{3+y-1}{3-1} \theta^9 (1 - \theta)^3 = 55 \cdot \theta^9 (1 - \theta)^3.$$

For a frequentist, large values of Y are critical and the p-value of the test is

$$P(Y \geq 9 | H_0) = \sum_{y=9}^{\infty} \binom{3+y-1}{2} (1/2)^y (1/2)^3 = 0.0327.$$

We used the following identity here

$$\sum_{x=k}^{\infty} \binom{2+x}{2} \frac{1}{2^x} = \frac{8 + 5k + k^2}{2^k}.$$

The hypothesis H_0 is rejected, and this change in decision is not caused by observations.

According to the Likelihood Principle, all relevant information is in the likelihood $l(\theta) \propto \theta^9 (1 - \theta)^3$, and Bayesians could not agree more!

Edwards, Lindman, and Savage (1963, 193) note: The likelihood principle emphasized in Bayesian statistics implies, among other things, that the rules governing when data collection stops are irrelevant to data interpretation. It is entirely appropriate to collect data until a point has been proven or disproven, or until the data collector runs out of time, money, or patience.

6.2 The Bayesian Approach

Formally, the Bayesian approach to hypothesis testing is a special case of the model comparison results to be discussed later. The Bayesian approach just computes the posterior distribution of each hypothesis. By Bayes

$$P(H_i | y) = \frac{p(y | H_i) P(H_i)}{p(y)}, \text{ for } i = 0, 1$$

where $P(H_i)$ is the prior probability of H_i ,

$$p(y | H_i) = \int_{\theta \in \Theta_i} p(y | \theta) p(\theta | H_i) d\theta$$

is the marginal likelihood under H_i , $p(\theta | H_i)$ is the parameter prior under H_i , and

$$p(y) = \sum_{i=0,1} p(y | H_i) P(H_i).$$

If the hypotheses are mutually exclusive, $P(H_0) = 1 - P(H_1)$.

The posterior *odds* of the null to the alternative is

$$\text{Odds}_{0,1} = \frac{P(H_0 | y)}{P(H_1 | y)} = \frac{p(y | H_0) P(H_0)}{p(y | H_1) P(H_1)}.$$

The odds ratio updates the prior odds, $P(H_0) / P(H_1)$, using the Bayes Factor,

$$\text{BF}_{0,1} = \frac{p(y | H_0)}{p(y | H_1)}.$$

With exhaustive competing hypotheses, $P(H_0 | y)$ simplifies to

$$P(H_0 | y) = \left(1 + (\text{BF}_{0,1})^{-1} \frac{(1 - P(H_0))}{P(H_0)} \right)^{-1},$$

and with equal prior probability, $P(H_0 | y) = \left(1 + (\text{BF}_{0,1})^{-1} \right)^{-1}$. Both Bayes factors and posterior probabilities can be used for comparing hypotheses. Jeffreys (1961) advocated using Bayes factors, and provided a scale for measuring the strength of evidence that was given earlier. Bayes factors merely indicate that the null hypothesis is more likely if $\text{BF}_{0,1} > 1$, $p(y | H_0) > p(y | H_1)$. The Bayesian approach merely compares density ordinates of $p(y | H_0)$ and $p(y | H_1)$, which mechanically involves plugging in the observed data into the functional form of the marginal likelihood.

For a point null, $H_0 : \theta = \theta_0$, the parameter prior is $p(\theta | H_0) = \delta_{\theta_0}(\theta)$ (a Dirac mass at θ_0), which implies that

$$p(y | H_0) = \int p(y | \theta_0) p(\theta | H_0) d\theta = p(y | \theta_0).$$

With a general alternative, $H_1 : \theta \neq \theta_0$, the probability of the null is

$$P(\theta = \theta_0 | y) = \frac{p(y | \theta_0) P(H_0)}{p(y | \theta_0) P(H_0) + (1 - P(H_0)) \int_{\Theta} p(y | \theta, H_1) p(\theta | H_1) d\theta},$$

where $p(\theta | H_1)$ is the parameter prior under the alternative. This formula will be used below.

Bayes factors and posterior null probabilities measure the relative weight of evidence of the hypotheses. Traditional hypothesis testing involves an additional decision or action: to accept or reject the null hypothesis. For Bayesians, this typically requires some statement of the utility/loss that codifies the benefits/costs of making a correct or incorrect decision. The simplest situation occurs if one assumes a zero loss of making a correct decision. The loss incurred when accepting the null (alternative) when the alternative is true (false) is $L(d_0 | H_1)$ and $L(d_1 | H_0)$, respectively.

The Bayesian will accept or reject based on the posterior expected loss. If the expected loss of accepting the null is less than the alternative, the rational decision maker will accept the null. The posterior loss of accepting the null is

$$\mathbb{E}[\mathcal{L} | d_0, y] = L(d_0 | H_0) P(H_0 | y) + L(d_0 | H_1) P(H_1 | y) = L(d_0 | H_1) P(H_1 | y),$$

since the loss of making a correct decision, $L(d_0 | H_0)$, is zero. Similarly,

$$\mathbb{E}[\mathcal{L} | d_1, y] = L(d_1 | H_0) P(H_0 | y) + L(d_1 | H_1) P(H_1 | y) = L(d_1 | H_0) P(H_0 | y).$$

Thus, the null is accepted if

$$\mathbb{E}[\mathcal{L} | d_0, y] < \mathbb{E}[\mathcal{L} | d_1, y] \iff L(d_0 | H_1) P(H_1 | y) < L(d_1 | H_0) P(H_0 | y),$$

which further simplifies to

$$\frac{L(d_0 | H_1)}{L(d_1 | H_0)} < \frac{P(H_0 | y)}{P(H_1 | y)}.$$

In the case of equal losses, this simplifies to accept the null if $P(H_1 | y) < P(H_0 | y)$. One advantage of Bayes procedures is that the resulting estimators and decisions are always admissible.

Example 6.2 (Enigma Code Breaking). Consider an alphabet of $A = 26$ letters. Let x and y be two transmitted messages of length T . We want to determine if they were encoded by the same Enigma machine setting (H_1) or by different/random settings (H_0).

To compute the Bayes factor, we compare the likelihood of the observed pair (x, y) under each hypothesis:

$$P(x, y | H_0) \text{ and } P(x, y | H_1).$$

Under H_0 (different settings), the two messages are effectively independent random sequences. The probability of any specific pair of letters is $(1/A)^2$, so for length T :

$$P(x, y | H_0) = \prod_{i=1}^T \left(\frac{1}{A}\right)^2 = \left(\frac{1}{A}\right)^{2T}.$$

Under H_1 (same setting), the messages are correlated. Specifically, if the letters at position i are the same ($x_i = y_i$), it implies a ‘match’. The probability of a match, denoted by m , depends on the language’s letter frequencies p_t (for English, $m = \sum p_t^2 \approx 0.066$ or about $2/26$). If they don’t match, the probability is distributed among the remaining pairs. Thus:

$$P(x_i, y_i | H_1) = \begin{cases} \frac{m}{A} & \text{if } x_i = y_i \text{ (match)} \\ \frac{1-m}{A(A-1)} & \text{if } x_i \neq y_i \text{ (mismatch)} \end{cases}$$

The term $1/A$ appears because we approximate the marginal probability of x_i as uniform, but the conditional probability $P(y_i|x_i)$ is boosted to m if $x_i = y_i$.

The log Bayes factor is the sum of contributions from matches (M) and mismatches (N):

$$\begin{aligned} \ln \frac{P(x, y | H_1)}{P(x, y | H_0)} &= M \ln \frac{m/A}{1/A^2} + N \ln \frac{(1-m)/A(A-1)}{1/A^2} \\ &= M \ln(mA) + N \ln \frac{(1-m)A}{A-1} \end{aligned}$$

Substituting values for English ($A = 26, m \approx 0.066$): The first term (match) adds $\ln(0.066 \times 26) \approx 0.54$. The second term (mismatch) subtracts $\ln(\frac{0.934 \times 26}{25}) \approx -0.01$. In base 10 (decibans), a match provides roughly 2.3 decibans of evidence, while a mismatch provides a slight penalty.

Example: With $T = 51$ letters, suppose we observe $M = 4$ matches and $N = 47$ mismatches. This yields:

$$4 \times 2.3 - 47 \times 0.03 \approx 9.2 - 1.41 = 7.79 \text{ decibans.}$$

This corresponds to a Bayes factor of roughly $10^{0.78} \approx 6$, providing evidence for H_1 .

How long a sequence do you need to look at? Calculate the expected log odds. Turing and Good figured you needed sequences of about length 400. Can also look at doubles and triples.

Example 6.3 (Dice and Odds Updating.). Suppose that you wish to assess whether a die is loaded or not. $H_0 : p = 1/6$ vs $H_1 : p = 1/5$. How will the evidence accumulate in each case?

Let $x = \# 6$'s and $y = \#$ non-6's. Then $x + y = n$. The posterior odds will update via the likelihood ratio (a.k.a. Bayes factor) as

$$\begin{aligned} O(H_0|D) &= \left(\frac{1/6}{1/5}\right)^x \left(\frac{5/6}{4/5}\right)^y O(H_0) \\ &= \left(\frac{5}{6}\right)^x \left(\frac{25}{24}\right)^y O(H_0) \end{aligned}$$

Under $H_0 : p = 1/6$ we can replace the data with the empirical cdf (a.k.a. $x/n = 1/6$) and similarly under H_1 we have $x/n = 1/5$.

Hence, we have

$$\frac{O(H_0|D)}{O(H_0)} \approx \left\{ \left(\frac{5}{6}\right)^{1/6} \left(\frac{25}{24}\right)^{5/6} \right\}^n = (1.00364)^n = 10^{0.00158n}$$

Hence, on a deciban scale (ten times the log-base-10 likelihood ratio, a term coined by I.J. Good), evidence accumulates at rate 0.00158 in favor of H_0 .

Under H_1 , we have

$$\frac{O(H_0|D)}{O(H_1)} \approx \left\{ \left(\frac{5}{6}\right)^{1/5} \left(\frac{25}{24}\right)^{4/5} \right\}^n = (0.9962)^n = 10^{-0.00165n}$$

Hence, on a deciban scale (IJ Good), evidence accumulates at rate 0.00165 against H_0 .

The Chernoff-Stein information lemma formalises this (T. M. Cover and Thomas 2006).

Example 6.4 (Signal Transmission). Suppose that the random variable X is transmitted over a noisy communication channel. Assume that the received signal is given by

$$Y = X + W,$$

where $W \sim N(0, \sigma^2)$ is independent of X . Suppose that $X = 1$ with probability p , and $X = -1$ with probability $1 - p$. The goal is to decide between $X = 1$ and $X = -1$ by observing the random variable Y . We will assume symmetric loss and will accept the hypothesis with the higher posterior probability. This is also sometimes called the maximum a posteriori (MAP) test.

We assume that $H_0 : X = 1$, thus $Y | H_0 \sim N(1, \sigma^2)$, and $Y | H_1 \sim N(-1, \sigma^2)$. The Bayes factor is simply the likelihood ratio

$$\frac{p(y | H_0)}{p(y | H_1)} = \exp\left(\frac{2y}{\sigma^2}\right).$$

The prior odds are $p/(1-p)$, thus the posterior odds are

$$\exp\left(\frac{2y}{\sigma^2}\right) \frac{p}{1-p}.$$

We choose H_0 (true X is 1), if the posterior odds are greater than 1, i.e.,

$$y > \frac{\sigma^2}{2} \log\left(\frac{1-p}{p}\right) = c.$$

Further, we can calculate the error probabilities of our test.

$$p(d_1 | H_0) = P(Y < c | X = 1) = \Phi\left(\frac{c-1}{\sigma}\right),$$

and

$$p(d_0 | H_1) = P(Y > c | X = -1) = 1 - \Phi\left(\frac{c+1}{\sigma}\right).$$

Let's plot the total error rate as a function of p and assuming $\sigma = 0.2$

$$P_e = p(d_1 | H_0)(1-p) + p(d_0 | H_1)p$$

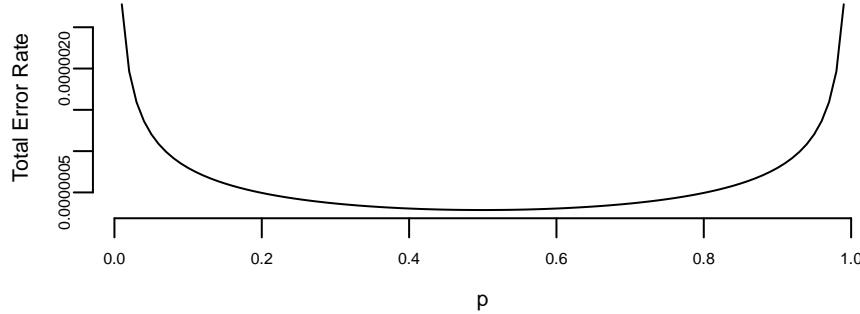


Figure 6.1: Total error rate as a function of p for $\sigma = 0.2$

Figure 6.1 shows the resulting Bayes (MAP) error rate as the prior probability p varies. As expected from the symmetry of the model, the curve is symmetric around $p = 1/2$: when the two hypotheses are equally likely, the decision threshold c is closest to 0 and the total probability of error is minimized. As p approaches 0 or 1, the threshold shifts to favor the more likely hypothesis, which reduces one type of mistake but increases the other, leading to a higher overall error rate.

Example 6.5 (Hockey: Hypothesis Testing for Normal Mean). The general manager of Washington Capitals (an NHL hockey team) thinks that their star

center player Evgeny Kuznetsov is underperforming and is thinking of trading him to a different team. He uses the number of goals per season as a metric of performance. He knows that historically, a top forward scores on average 30 goals per season with a standard deviation of 5, $\theta \sim N(30, 25)$. In the 2022-2023 season Kuznetsov scored 12 goals. For the number of goals $X | \theta$ he uses normal likelihood $N(\theta, 36)$. Kuznetsov's performance was not stable over the years, thus the high variance in the likelihood. Thus, the posterior is $N(23, 15)$.

```
sigma2 <- 36
sigma02 <- 25
mu <- 30
y <- 12
k <- sigma02 + sigma2
mu1 <- sigma2 / k * mu + sigma02 / k * y
sigma21 <- sigma2 * sigma02 / k
mu1
## 23
sigma21
## 15
```

The manager thinks that Kuznetsov simply had a bad year and his true performance is at least 24 goals per season $H_0 : \theta \geq 24$, $H_1 : \theta < 24$. The posterior probability of the H_0 hypothesis is

```
a <- 1 - pnorm(24, mu1, sqrt(sigma21))
a
## 0.36
```

It is less than 1/2, only 36%. Thus, we should reject the null hypothesis. The posterior odds in favor of the null hypothesis are

```
a / (1 - a)
## 0.56
```

If underestimating (and trading) Kuznetsov is two times more costly than overestimating him (fans will be upset and team spirit might be affected), that is $L(d_1 | H_0) = 2L(d_0 | H_1)$, then we should accept the null when posterior odds are greater than 1/2. This is the case here, 0.55 is greater than 1/2. The posterior odds are in favor of the null hypothesis. Thus, the manager should not trade Kuznetsov.

Kuznetsov was traded to Carolina Hurricanes towards the end of the 2023-2024 season.

Notice, when we try to evaluate a newcomer to the league, we use the prior probability of $\theta \geq 24$:

```
a <- 1 - pnorm(24, mu, sqrt(sigma02))
print(a)
## 0.88
a / (1 - a)
## 7.7
```

Thus, the prior odds in favor of H_0 are 7.7.

Example 6.6 (Hypothesis Testing for Normal Mean: Two-Sided Test). In the case of two sided test, we are interested in testing

- $H_0 : \theta = \theta_0, p(\theta | H_0) = \delta_{\theta_0}(\theta)$
- $H_1 : \theta \neq \theta_0, p(\theta | H_1) = N(\theta_0, \sigma^2/n_0)$

Where n is the sample size and σ^2 is the variance (known) of the population. Observed samples are $Y = (y_1, y_2, \dots, y_n)$ with

$$y_i | \theta, \sigma^2 \sim N(\theta, \sigma^2).$$

The Bayes factor can be calculated analytically

$$\begin{aligned} BF_{0,1} &= \frac{p(Y | \theta = \theta_0, \sigma^2)}{\int p(Y | \theta, \sigma^2) p(\theta | \theta_0, n_0, \sigma^2) d\theta} \\ \int p(Y | \theta, \sigma^2) p(\theta | \theta_0, n_0, \sigma^2) d\theta &= \frac{\sqrt{n_0} \exp \left\{ -\frac{n_0(\theta_0 - \bar{y})^2}{2(n_0 + n)\sigma^2} \right\}}{\sqrt{2\pi}\sigma^2 \sqrt{\frac{n_0+n}{\sigma^2}}} \\ p(Y | \theta = \theta_0, \sigma^2) &= \frac{\exp \left\{ -\frac{(\bar{y} - \theta_0)^2}{2\sigma^2} \right\}}{\sqrt{2\pi}\sigma} \end{aligned}$$

Thus, the Bayes factor is

$$\begin{aligned} BF_{0,1} &= \frac{\sigma \sqrt{\frac{n_0+n}{\sigma^2}} e^{-\frac{(\theta_0 - \bar{y})^2}{2(n_0+n)\sigma^2}}}{\sqrt{n_0}} \\ BF_{0,1} &= \left(\frac{n+n_0}{n_0} \right)^{1/2} \exp \left\{ -\frac{1}{2} \frac{n}{n+n_0} Z^2 \right\} \\ Z &= \frac{(\bar{Y} - \theta_0)}{\sigma/\sqrt{n}} \end{aligned}$$

One way to interpret the scaling factor n_0 is to look at the standard effect size

$$\delta = \frac{\theta - \theta_0}{\sigma}.$$

The prior of the standard effect size is

$$\delta \mid H_1 \sim N(0, 1/n_0).$$

This allows us to think about a standardized effect independent of the units of the problem.

Let's consider now example of Argon discovery.

```

air <- c(2.31017, 2.30986, 2.31010, 2.31001, 2.31024, 2.31010,
       ↵ 2.31028, 2.31028)
decomp <- c(2.30143, 2.29890, 2.29816, 2.30182, 2.29869,
           ↵ 2.29940, 2.29849, 2.29889)

```

Our null hypothesis is that the mean of the difference equals to zero. We assume that measurements made in the lab have normal errors, this the normal likelihood. We empirically calculate the standard deviation of our likelihood. The Bayes factor is

We have extremely strong evidence in favor $H_1 : \theta \neq 0$ hypothesis. The posterior probability of the alternative hypothesis is numerically 1!

```
a <- 1 / (1 + BF)  
a  
## 1
```

Example 6.7 (Hypothesis Testing for Proportions). Let's look again at the effectiveness of Google's new search algorithm. We measure effectiveness by the number of users who clicked on one of the search results. As users send

the search requests, they will be randomly processed with Algo 1 or Algo 2. We wait until 2500 search requests were processed by each of the algorithms and calculate the following table based on how often people clicked through

	Algo1	Algo2
success	1755	1818
failure	745	682
total	2500	2500

Here we assume binomial likelihood and use conjugate beta prior, for mathematical convenience. We are putting independent beta priors on the click-through rates of the two algorithms, $p_1 \sim Beta(\alpha_1, \beta_1)$ and $p_2 \sim Beta(\alpha_2, \beta_2)$. The posterior for p_1 and p_2 are independent Beta distributions

$$p(p_1, p_2 | y) \propto p_1^{\alpha_1+1755-1} (1-p_1)^{\beta_1+745-1} \times p_2^{\alpha_2+1818-1} (1-p_2)^{\beta_2+682-1}.$$

The easiest way to explore this posterior is via Monte Carlo simulation of the posterior.

```
set.seed(92) # Kuzy
y1 <- 1755
n1 <- 2500
alpha1 <- 1
beta1 <- 1
y2 <- 1818
n2 <- 2500
alpha2 <- 1
beta2 <- 1
m <- 10000
p1 <- rbeta(m, y1 + alpha1, n1 - y1 + beta1)
p2 <- rbeta(m, y2 + alpha2, n2 - y2 + beta2)
rd <- p2 - p1
plot(density(rd), xlab = "p2 - p1", ylab = "Density", lwd = 3)
q <- quantile(rd, c(.05, .95))
print(q)
##      5%     95%
## 0.0037 0.0465
abline(v = q, col = "red", lwd = 2)
```

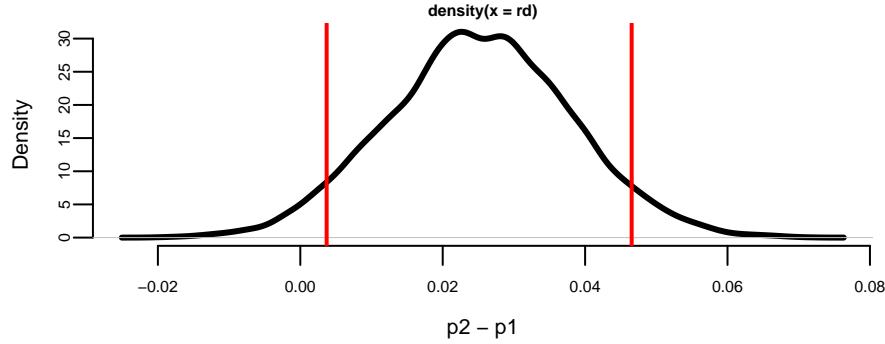


Figure 6.2: Posterior Difference in Click-Through Rates

6.3 Interval Estimation: Credible Sets

The interval estimators of model parameters are called credible sets. If we use the posterior measure to assess the credibility, the credible set is a set of parameter values that are consistent with the data and gives us a natural way to measure the uncertainty of the parameter estimate.

Those who are familiar with the concept of classical confidence intervals (CI's) often make an error by stating that the probability that the CI interval $[L, U]$ contains parameter θ is $1 - \alpha$. The right statement seems convoluted, one needs to generate data from such model many times and for each data set to exhibit the CI. Now, the proportion of CI's covering the unknown parameter is "tends to" $1 - \alpha$. Bayesian interpretation of a credible set C is natural: The probability of a parameter belonging to the set C is $1 - \alpha$. A formal definition follows. Assume the set C is a subset of domain of the parameter Θ . Then, C is credible set with credibility $(1 - \alpha) \cdot 100\%$ if

$$p(\theta \in C | y) = \int_C p(\theta | y) d\theta \geq 1 - \alpha.$$

If the posterior is discrete, then the integral becomes sum (counting measure) and

$$p(\theta \in C | y) = \sum_{\theta_i \in C} p(\theta_i | y) \geq 1 - \alpha.$$

This is the definition of a $(1 - \alpha)100\%$ credible set, and of course for a given posterior function such set is not unique.

For a given credibility level $(1 - \alpha)100\%$, the shortest credible set is of interest. To minimize size the sets should correspond to highest posterior probability (density) areas. Thus the acronym HPD.

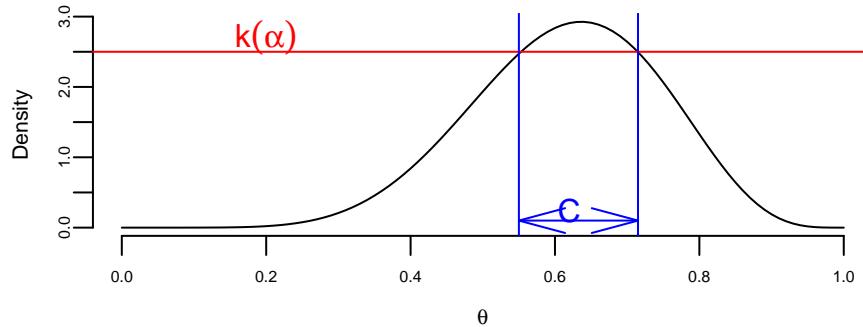
Definition 6.1 (Highest Posterior Density (HPD) Credible Set). The $(1 - \alpha)$ 100% HPD credible set for parameter θ is a set $C \subset \Theta$ of the form

$$C = \{\theta \in \Theta : p(\theta | y) \geq k(\alpha)\},$$

where $k(\alpha)$ is the largest value such that

$$P(\theta \in C | y) = \int_C p(\theta | y) d\theta \geq 1 - \alpha.$$

Geometrically, if the posterior density is cut by a horizontal line at the height $k(\alpha)$, the set C is the projection on the θ axis of the region where the posterior density lies above the line.



Lemma 6.1. *The HPD set C minimizes the size among all sets $D \subset \Theta$ for which*

$$P(\theta \in D) = 1 - \alpha.$$

Proof. The proof is essentially a special case of Neyman-Pearson lemma. If $I_C(\theta) = 1(\theta \in C)$ and $I_D(\theta) = 1(\theta \in D)$, then the key observation is

$$(p(\theta | y) - k(\alpha))(I_C(\theta) - I_D(\theta)) \geq 0.$$

Indeed, for θ 's in $C \cap D$ and $(C \cup D)^c$, the factor $I_C(\theta) - I_D(\theta) = 0$. If $\theta \in C \cap D^c$, then $I_C(\theta) - I_D(\theta) = 1$ and $p(\theta | y) - k(\alpha) \geq 0$. If, on the other hand, $\theta \in D \cap C^c$, then $I_C(\theta) - I_D(\theta) = -1$ and $p(\theta | y) - k(\alpha) \leq 0$. Thus,

$$\int_{\Theta} (p(\theta | y) - k(\alpha))(I_C(\theta) - I_D(\theta)) d\theta \geq 0.$$

The statement of the theorem now follows from the chain of inequalities,

$$\int_C (p(\theta | y) - k(\alpha)) d\theta \geq \int_D (p(\theta | y) - k(\alpha)) d\theta$$

$$(1 - \alpha) - k(\alpha)\text{size}(C) \geq (1 - \alpha) - k(\alpha)\text{size}(D)$$

$$\text{size}(C) \leq \text{size}(D).$$

The size of a set is simply its total length if the parameter space θ is one dimensional, total area, if θ is two dimensional, and so on. \square

Note, when the distribution $p(\theta | y)$ is unimodal and symmetric using quantiles of the posterior distribution is a good way to obtain the HPD set.

An equal-tailed interval (also called a central interval) of confidence level

$$I_\alpha = [q_{\alpha/2}, q_{1-\alpha/2}],$$

here q 's are the quantiles of the posterior distribution. This is an interval on whose both right and left side lies $(1 - \alpha/2)100\%$ of the probability mass of the posterior distribution; hence the name equal-tailed interval.

Usually, when a credible interval is mentioned without specifying which type of the credible interval it is, an equal-tailed interval is meant.

However, unless the posterior distribution is unimodal and symmetric, there are points outside of the equal-tailed credible interval having a higher posterior density than some points of the interval. If we want to choose the credible interval so that this not happen, we can do it by using the highest posterior density criterion for choosing it.

Example 6.8 (Cauchy.). Assume that the observed samples

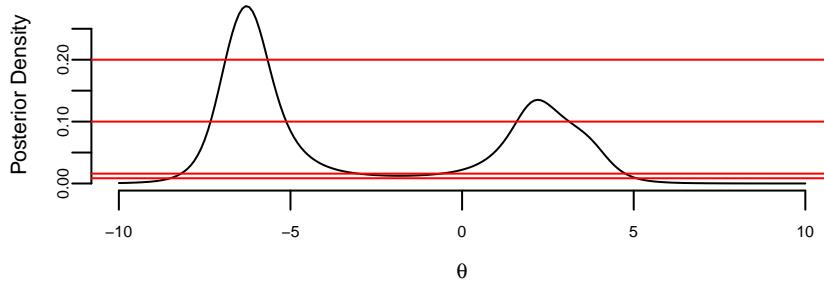
```
y <- c(2, -7, 4, -6)
```

come from Cauchy distribution. The likelihood is

$$p(y | \theta, \gamma) = \frac{1}{\pi\gamma} \prod_{i=1}^4 \frac{1}{1 + \left(\frac{y_i - \theta}{\gamma}\right)^2}.$$

We assume unknown location parameter θ and scale parameter $\gamma = 1$. For the flat prior $\pi(\theta) = 1$, the posterior is proportional to the likelihood.

```
par(mar = c(4, 4, 0, 0), bty = "n")
lhood <- function(theta) 1 / prod(1 + (y - theta)^2)
theta <- seq(-10, 10, 0.1)
post <- sapply(theta, lhood)
post <- 10 * post / sum(post)
plot(theta, post, type = "l", xlab = expression(theta), ylab =
  "Posterior Density")
abline(h = c(0.008475, 0.0159, 0.1, 0.2), col = "red")
```



The four horizontal lines correspond to four credible sets

k	C	$P(\theta \in C y)$
0.008475	[-8.498, 5.077]	99%
0.0159	[-8.189, -3.022] \cup [-0.615, 4.755]	95%
0.1	[-7.328, -5.124] \cup [1.591, 3.120]	64.2%
0.2	[-6.893, -5.667]	31.2%

Notice that for $k = 0.0159$ and $k = 0.1$ the credible set is not a compact. This shows that two separate intervals “clash” for the ownership of θ and this is a useful information. This non-compactness can also point out that the prior is not agreeing with the data. There is no frequentist counterpart for the CI for θ in the above model.

6.4 Alternative Approaches

The two main alternatives to the Bayesian approach are significance testing using p -values, developed by Ronald Fisher, and the Neyman-Pearson approach.

6.4.1 Significance testing using p-values

Fisher’s approach posits a test statistic, $T(y)$, based on the observed data. In Fisher’s mind, if the value of the statistic was highly unlikely to have occurred under H_0 , then the H_0 should be rejected. Formally, the p -value is defined

as

$$p = P [T(Y) > T(y) \mid H_0],$$

where y is the observed sample and $Y = (Y_1, \dots, Y_T)$ is a random sample generated from model $p(Y \mid H_0)$, that is, the null distribution of the test-statistic in repeated samples. Thus, the p -value is the probability that a data set would generate a more extreme statistic under the null hypothesis, and not the probability of the null, conditional on the data.

The testing procedure is simple. Fisher (1946, p. 80) argues that: If P (the p -value) is between* 0.1 and 0.9, there is certainly no reason to suspect the hypothesis tested. If it is below 0.02, it is strongly indicated that the hypothesis fails to account for the whole of the facts. We shall not be astray if we draw a line at 0.05 and consider that higher values of χ^2 indicate a real discrepancy. Defining α as the significance level, the tests rejects H_0 if $p < \alpha$. Fisher advocated a fixed significance level of 5%, based largely that 5% is roughly the tail area of a mean zero normal distribution more than two standard deviations from 0, indicating a statistically significant departure. In practice, testing with p -values involves identifying a critical value, t_α , and rejecting the null if the observed statistic $t(y)$ is more extreme than t_α . For example, for a significance test of the sample mean, $t(y) = (\bar{y} - \theta_0) / se(\bar{y})$, where $se(\bar{y})$ is the standard error of \bar{y} ; the 5% critical value is 1.96; and Fisher would reject the null if $t(y) > t_\alpha$.

Fisher interpreted the p -value as the weight or measure of evidence of the null hypothesis. The alternative hypothesis is noticeable in its absence in Fisher's approach. Fisher largely rejected the consideration of alternatives, believing that researchers should weigh the evidence or draw conclusions about the observed data rather than making decisions such as accepting or rejecting hypotheses based on it.

There are a number of issues with Fisher's approach. The first and most obvious criticism is that it is possible to reject the null, when the alternative hypothesis is less likely. This is an inherent problem in using population tail probabilities—essentially rare events. Just because a rare event has occurred does not mean the null is incorrect, unless there is a more likely alternative. This situation often arises in court cases, where a rare event like a murder has occurred. Decisions based on p -values generates a problem called prosecutor's Fallacy, which is discussed below. Second, Fisher's approach relies on population properties (the distribution of the statistic under the null) that would only be revealed in repeated samples or asymptotically. Thus, the testing procedure relies on data that is not yet seen, a violation of what is known as the likelihood principle. As noted by Jeffreys' (1939, pp. 315-316): “*What the use of P implies, therefore, is that a hypothesis that may be true may be rejected because it has not predicted observable data that have not occurred. This seems a remarkable procedure*”

Third, Fisher is agnostic regarding the source of the test statistics, providing

no discussion of how the researcher decides to focus on one test statistic over another. In some simple models, the distribution of properly scaled sufficient statistics provides natural test statistics (e.g., the t -test). In more complicated models, Fisher is silent on the sources. In many cases, there are numerous test statistics (e.g., testing for normality), and test choice is clearly subjective. For example, in Generalized Method of Moments (GMM) tests, the choice of test moments is clearly a subjective choice. Finally, from a practical perspective, p -values have a serious deficiency: tests using p -values often appear to give the wrong answer, in the sense that they provide a highly misleading impression of the weight of evidence in many samples. A number of examples of this will be given below, but in all cases, Fisher's approach tends to over-reject the null hypotheses.

6.4.2 Bayes vs P-value

The fundamental difference between the Bayesian and frequentist approaches can be summarized by how they quantify evidence. As R.A. Fisher famously wrote: "... for the one chance in a million will undoubtedly occur, with no less and no more than its appropriate frequency, however surprised we may be that it should occur to us ...". This quote highlights that rare events do happen under the null hypothesis.

The Bayesian evidence is quantified by the Bayes Factor:

$$BF = \frac{P(D|H_0)}{P(D|H_1)}$$

If we consider the data definition used in frequentist testing, where $D = \{T(x) > t\}$, then the numerator $P(D|H_0)$ corresponds precisely to the **p-value**. The crucial limitation of the p-value is that it considers only the numerator. Even if this probability is small (suggesting the data is rare under the null), the denominator $P(D|H_1)$ —the probability of observing such data under the alternative—might be **even smaller!**

In a case with mutually exhaustive hypotheses ($P(H_0) + P(H_1) = 1$), it is perfectly possible for the p-value to be small (e.g., 0.05), yet for the Bayes Factor to be greater than 1 ($BF > 1$), indicating that the evidence actually favors H_0 . This highlights the danger of assessing hypotheses in isolation rather than comparing them relative to one another.

Example 6.9 (Ulcer Treatment Clinical Trial). One of the earliest applications of Bayesian methods to clinical trials was presented by Novick and Grizzle (1965), who analyzed data from an ongoing experiment comparing four operative treatments for duodenal ulcers. Doctors assessed patient outcomes as Excellent, Fair, or Death. The data, collected sequentially over the course of the trial, are shown below:

Table 6.4: Outcomes by treatment for duodenal ulcer surgery

Treatment	Excellent	Fair	Death	Total
A	76	17	7	100
B	89	10	1	100
C	86	13	1	100
D	88	9	3	100

When I present this data to students and ask which treatment they would prefer, most choose Treatment B. The reasoning is intuitive: Treatment B has the highest number of excellent outcomes (89) and the lowest death rate (only 1 death, tied with C). Treatment A, despite being listed first, has only 76 excellent outcomes and 7 deaths, the worst performance on both metrics. The students' intuition is correct, but can we quantify how confident we should be that B is truly better than A?

A classical chi-square test of homogeneity across treatments yields a p-value greater than 0.05, leading to the conclusion that we *cannot reject* the null hypothesis that all treatments are equally effective at the 5% significance level. This non-rejection is often misinterpreted as evidence that the treatments are equivalent.

But consider what the p-value actually measures: $p = P(D | H_0)$, the probability of observing data at least as extreme as what we saw, assuming the null hypothesis is true. The critical insight is that $P(D | H_1)$, the probability of the data under any specific alternative, can be much smaller than $P(D | H_0)$. A small p-value does not mean the alternative is more likely; it only means the observed data would be rare under the null. This is the essence of the Bayesian critique: inference should be *relative*, comparing the evidence for different hypotheses, not absolute.

The Bayesian approach directly addresses the question of interest: given the observed data, what is the probability that one treatment is better than another? Let p_i denote the death rate under treatment i . Using independent Beta priors for each treatment's death rate, say $p_i \sim \text{Beta}(1, 1)$ (uniform), the posteriors after observing the data are:

$$p_A | \text{data} \sim \text{Beta}(1 + 7, 1 + 93) = \text{Beta}(8, 94)$$

$$p_B | \text{data} \sim \text{Beta}(1 + 1, 1 + 99) = \text{Beta}(2, 100)$$

```
# Posterior distributions for death rates
# Treatment A: 7 deaths out of 100
# Treatment B: 1 death out of 100
alpha_A <- 1 + 7
```

```

beta_A <- 1 + 93
alpha_B <- 1 + 1
beta_B <- 1 + 99

# Monte Carlo estimate of P(p_A > p_B | data)
set.seed(123)
n_sims <- 100000
p_A_samples <- rbeta(n_sims, alpha_A, beta_A)
p_B_samples <- rbeta(n_sims, alpha_B, beta_B)

prob_A_worse <- mean(p_A_samples > p_B_samples)
cat("P(death rate A > death rate B | data) =",
    ↪ round(prob_A_worse, 3), "\n")
## P(death rate A > death rate B | data) = 0.98

```

The posterior probability that Treatment A has a higher death rate than Treatment B is approximately 0.98. This is a direct, interpretable answer to the clinical question: there is a 98% probability that patients receiving Treatment A face higher mortality risk than those receiving Treatment B.

The contrast with the frequentist conclusion is striking. The chi-square test fails to reject equality at the 5% level, which many would interpret as “no difference.” The Bayesian analysis reveals that we can be 98% confident that Treatment A is worse than Treatment B in terms of mortality. The difference arises because the Bayesian approach compares hypotheses directly, while the p-value only measures how surprising the data would be under the null.

```

# Frequentist chi-square test of homogeneity
# Construct the contingency table (Excellent, Fair, Death) x
#   (Treatment A, B, C, D)
outcome_matrix <- matrix(c(
  76, 17, 7, # Treatment A
  89, 10, 1, # Treatment B
  86, 13, 1, # Treatment C
  88, 9, 3 # Treatment D
), nrow = 4, byrow = TRUE)
rownames(outcome_matrix) <- c("A", "B", "C", "D")
colnames(outcome_matrix) <- c("Excellent", "Fair", "Death")

# Chi-square test
chisq_result <- suppressWarnings(chisq.test(outcome_matrix))
cat("Chi-square statistic:", round(chisq_result$statistic, 2),
    ↪ "\n")
## Chi-square statistic: 12
cat("Degrees of freedom:", chisq_result$parameter, "\n")

```

```

## Degrees of freedom: 6
cat("P-value:", round(chisq_result$p.value, 3), "\n")
## P-value: 0.053

# Fisher's exact test for just the death column (A vs B)
death_table <- matrix(c(7, 93, 1, 99), nrow = 2, byrow = TRUE)
rownames(death_table) <- c("A", "B")
colnames(death_table) <- c("Death", "Survival")
fisher_result <- fisher.test(death_table, alternative =
  "greater")
cat("\nFisher's exact test (A vs B, deaths only):\n")
##
## Fisher's exact test (A vs B, deaths only):
cat("P-value:", round(fisher_result$p.value, 3), "\n")
## P-value: 0.032

```

The chi-square test yields a p-value of 0.053, which exceeds the conventional 0.05 threshold. Even Fisher's exact test comparing only the death rates between Treatments A and B gives a p-value around 0.03, which is only marginally significant and provides no sense of the magnitude of the difference or our confidence in it. The Bayesian approach, by contrast, tells us directly that there is a 98% probability that Treatment A has a higher death rate.

This example illustrates a fundamental principle: *Bayesian inference is relative*. We do not ask whether the data are unlikely in some absolute sense; we ask which hypothesis better explains the data. There are no absolutes in Bayesian inference, only comparisons. A treatment is not declared “effective” or “ineffective” in isolation; it is compared to alternatives, with uncertainty fully quantified.

```

p_grid <- seq(0, 0.2, length.out = 500)
plot(p_grid, dbeta(p_grid, alpha_A, beta_A),
  type = "l", col = "red", lwd = 2,
  xlab = "Death rate", ylab = "Posterior density", ylim = c(0,
    50)
)
lines(p_grid, dbeta(p_grid, alpha_B, beta_B), col = "blue", lwd
  = 2)
legend("topright",
  legend = c("Treatment A", "Treatment B"),
  col = c("red", "blue"), lwd = 2, bty = "n"
)

# Add posterior means
abline(v = alpha_A / (alpha_A + beta_A), col = "red", lty = 2)

```

```
abline(v = alpha_B / (alpha_B + beta_B), col = "blue", lty = 2)
```

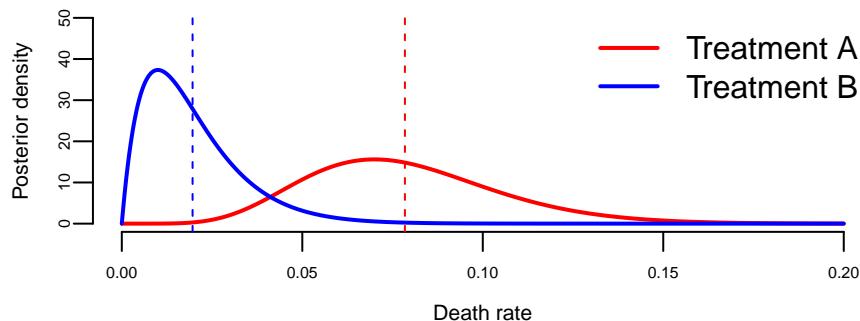


Figure 6.3: Posterior distributions of death rates

The posterior distributions show clear separation between the two treatments. Treatment A's death rate is concentrated around 7-8%, while Treatment B's is concentrated around 1-2%. The overlap is minimal, corresponding to our finding that $P(p_A > p_B | \text{data}) \approx 0.98$.

Prior Sensitivity Analysis

A natural concern with Bayesian analysis is whether the conclusions depend heavily on the choice of prior. We used a uniform Beta(1,1) prior, but what if we had used different priors? The table below shows how $P(p_A > p_B | \text{data})$ varies across several reasonable prior specifications:

Table 6.5: Sensitivity of posterior probability to prior specification

Prior	Prior Mean	$P(p_A > p_B \text{data})$
Uniform Beta(1,1)	0.50	0.98
Jeffreys Beta(0.5,0.5)	0.50	0.99
Skeptical Beta(2,20)	0.09	0.97
Optimistic Beta(1,10)	0.09	0.98
Strong Beta(5,50)	0.09	0.94

The results demonstrate remarkable robustness. Regardless of whether we use a uniform prior, the Jeffreys prior, or informative priors centered on different mortality rates, the posterior probability that Treatment A has a higher death rate than Treatment B remains above 0.93. This robustness occurs because the sample size (100 patients per treatment) is large enough that the likelihood generally dominates the prior. The data speak loudly, and the conclusion is not an artifact of our prior beliefs.

This sensitivity analysis is particularly important in clinical settings, where different stakeholders may have different prior beliefs about treatment efficacy. The fact that all reasonable priors lead to the same qualitative conclusion (strong evidence that A is worse than B) strengthens the case for preferring Treatment B.

6.4.3 Neyman-Pearson

The motivation for the Neyman-Pearson (NP) approach was W.S. Gosset, the famous Student who invented the t -test. In analyzing a hypothesis, Student argued that a hypothesis is not rejected unless an alternative is available that provides a more plausible explanation of the data, in which case. Mathematically, this suggests analyzing the likelihood ratio,

$$\mathcal{LR}_{0,1} = \frac{p(y | H_0)}{p(y | H_1)},$$

and rejecting the null in favor of the alternative when the likelihood ratio is small enough, $\mathcal{LR}_{0,1} < k$. This procedure conforms in spirit with the Bayesian approach.

The main problem was one of finding a value of the cut off parameter k . From the discussion above, by varying k , one varies the probabilities of type one and type two errors in the testing procedure. Neyman and Pearson (1933a) argued that the balance between Type I and II errors is subjective: “*how the balance (between the type I and II errors) should be struck must be left to the investigator*”. This approach, however, was not “objective”, and they then advocated fixing α , the probability of a type I error, in order to determine k . This led to their famous lemma:

Lemma 6.2 (Neyman-Pearson Lemma). *Consider the simple hypothesis test of $H_0 : \theta = \theta_0$ versus $H_1 : \theta = \theta_1$ and suppose that the null is rejected if $\mathcal{LR}_{0,1} < k_\alpha$, where k_α is chosen to fix the probability of a type I error at α :*

$$\alpha = P[y : \mathcal{LR}_{0,1} < k_\alpha | H_0].$$

Then, this test is the most powerful test of size α in the sense that any other test with greater power, must have a higher size.

In the case of composite hypothesis tests, parameter estimation is required under the alternative, which can be done via maximum likelihood, leading to the likelihood ratio

$$\mathcal{LR}_{0,1} = \frac{p(y | H_0)}{\sup_{\theta \in \Theta} p(y | \Theta)} = \frac{p(y | H_0)}{p(y | \hat{\theta})},$$

where $\hat{\theta}$ is the MLE. Because of this, $0 \leq \mathcal{LR}_{0,1} \leq 1$ for composite hypotheses. In multi-parameter cases, finding the distribution of the likelihood ratio is more difficult, requiring asymptotic approximations to calibrate k_α .

At first glance, the NP approach appears similar to the Bayesian approach, as it takes into account the likelihood ratio. However, like the p -value, the NP approach has a critical flaw. Neyman and Pearson fix the Type I error, and then minimizes the type II error. In many practical cases, α is set at 5% and the resulting β is often very small, close to 0. Why is this a reasonable procedure? Given the previous discussion, this is essentially a very strong prior over the relative benefits/costs of different types of errors. While these assumptions may be warranted in certain settings, it is difficult to a priori understand why this procedure would generically make sense. The next section highlights how the p -value and NP approaches can generate counterintuitive and even absurd results in standard settings.

6.5 Sequential Analysis

Sequential analysis represents one of the most natural applications of Bayesian reasoning, allowing researchers to evaluate evidence as it accumulates and make principled decisions about when sufficient information has been gathered. Unlike fixed-sample designs that predetermine the number of observations, sequential methods continuously update beliefs about hypotheses and can terminate data collection once a satisfactory conclusion is reached.

The foundations of sequential analysis were laid by Abraham Wald during World War II, with his development of the Sequential Probability Ratio Test (SPRT) (Wald 1945). Wald's work, later expanded in his book *Sequential Analysis* (Wald 1947), demonstrated that sequential procedures could reduce the expected sample size by up to 50% compared to fixed-sample tests while maintaining the same error rates. The key insight was that rather than collecting a predetermined number of observations and then analyzing them, one could examine the data after each observation and stop as soon as the evidence was sufficiently strong in either direction.

6.5.1 The Bayesian Framework for Sequential Testing

The Bayesian approach to sequential testing provides a coherent framework for deciding when to stop collecting data. Consider testing $H_0 : \theta \in \Theta_0$ versus

$H_1 : \theta \in \Theta_1$. After observing data y_1, y_2, \dots, y_n , the posterior odds are

$$\frac{P(H_0 | y_{1:n})}{P(H_1 | y_{1:n})} = \frac{p(y_{1:n} | H_0)}{p(y_{1:n} | H_1)} \cdot \frac{P(H_0)}{P(H_1)}.$$

A natural stopping rule is to continue sampling until the posterior probability of one hypothesis exceeds some threshold. For example, stop and accept H_0 if $P(H_0 | y_{1:n}) > 1 - \alpha$, or stop and accept H_1 if $P(H_1 | y_{1:n}) > 1 - \beta$. This approach has a compelling interpretation: we continue gathering evidence until we are sufficiently confident in our conclusion.

The Bayesian perspective offers a key advantage over frequentist sequential procedures: the posterior probability is always valid, regardless of when or why sampling stopped. As Edwards, Lindman, and Savage (1963) noted, the likelihood principle implies that the rules governing when data collection stops are irrelevant to data interpretation. This property, sometimes called *optional stopping*, means that Bayesian inference is immune to the criticism that plagues frequentist sequential analysis, where p-values become invalid if the stopping rule is not prespecified.

6.5.2 Wald's Sequential Probability Ratio Test

While Wald's SPRT was developed from a frequentist perspective, it has deep connections to Bayesian testing. For simple hypotheses $H_0 : \theta = \theta_0$ versus $H_1 : \theta = \theta_1$, the SPRT accumulates the likelihood ratio

$$\Lambda_n = \frac{p(y_{1:n} | \theta_1)}{p(y_{1:n} | \theta_0)} = \prod_{i=1}^n \frac{p(y_i | \theta_1)}{p(y_i | \theta_0)}$$

and stops when $\Lambda_n \leq A$ (accept H_0) or $\Lambda_n \geq B$ (accept H_1), where the boundaries A and B are chosen to achieve desired error rates.

This is precisely the Bayes factor for simple hypotheses. With equal prior probabilities, the SPRT stopping rule becomes: stop and accept H_0 when the posterior probability exceeds $B/(1 + B)$, or stop and accept H_1 when it exceeds $1/(1 + A)$. Wald proved that among all tests with the same error rates, the SPRT minimizes the expected sample size under both hypotheses, a remarkable optimality property.

6.5.3 Applications in Clinical Trials

Clinical trials represent perhaps the most important application of sequential analysis, where the ethical imperative to minimize patient exposure to inferior treatments aligns with the statistical goal of efficient inference. Peter Armitage

pioneered the application of sequential methods to clinical trials in the 1950s and 1960s (Armitage 1975), demonstrating how these methods could reduce trial duration and the number of patients receiving suboptimal treatments.

Modern clinical trial design frequently employs *group sequential designs*, which allow interim analyses at predetermined points during patient accrual. The key challenge is controlling the overall Type I error rate when multiple looks at the data are permitted. Frequentist approaches use spending functions to allocate alpha across interim analyses (DeMets and Lan 1994), while Bayesian approaches naturally handle multiple looks through the posterior probability framework.

The Bayesian approach to clinical trial design has been championed by Donald Berry, whose seminal work (D. A. Berry 1985) critiqued traditional hypothesis testing and advocated for a framework where sampling stops when the posterior probability that one treatment is superior exceeds a specified threshold. Berry's methodology, later expanded in S. M. Berry et al. (2010), integrates Bayesian decision theory into trial design and monitoring, enabling continuous assessment of accumulating data and facilitating decisions on early termination for efficacy or futility.

Berry's Bayesian Framework for Clinical Trials

Consider a clinical trial comparing a new treatment to a control, where the primary outcome is binary (success or failure). Let p_T denote the success probability under treatment and p_C under control. The quantity of interest is typically the treatment effect, which can be parameterized as the risk difference $\delta = p_T - p_C$, the relative risk p_T/p_C , or the odds ratio.

Berry's approach specifies conjugate beta priors for each success probability:

$$p_C \sim \text{Beta}(\alpha_C, \beta_C) \quad \text{and} \quad p_T \sim \text{Beta}(\alpha_T, \beta_T).$$

Non-informative priors correspond to $\alpha = \beta = 1$ (uniform) or $\alpha = \beta = 0.5$ (Jeffreys). After observing x_C successes in n_C control patients and x_T successes in n_T treated patients, the posteriors are

$$\begin{aligned} p_C \mid \text{data} &\sim \text{Beta}(\alpha_C + x_C, \beta_C + n_C - x_C) \\ p_T \mid \text{data} &\sim \text{Beta}(\alpha_T + x_T, \beta_T + n_T - x_T). \end{aligned}$$

The key quantity for decision-making is the posterior probability that treatment is superior:

$$P(p_T > p_C \mid \text{data}) = \int_0^1 \int_{p_C}^1 f(p_C \mid \text{data}) f(p_T \mid \text{data}) dp_T dp_C,$$

where $f(\cdot \mid \text{data})$ denotes the posterior density. This integral can be computed via Monte Carlo simulation by drawing samples from each posterior and computing the proportion where $p_T^{(s)} > p_C^{(s)}$.

Berry's stopping rules are defined in terms of posterior probability thresholds:

- *Efficacy stopping:* Stop and declare treatment effective if $P(p_T > p_C + \delta_{\min} \mid \text{data}) > \theta_E$, where δ_{\min} is the minimum clinically meaningful difference and θ_E is the efficacy threshold (e.g., 0.95 or 0.99).
- *Futility stopping:* Stop for futility if $P(p_T > p_C \mid \text{data}) < \theta_F$, where θ_F is the futility threshold (e.g., 0.05 or 0.10).

A more sophisticated approach uses *predictive probability of success* (PPoS), which accounts for future data that might be collected. The PPoS at interim analysis is the probability, given current data, that the trial will demonstrate efficacy if continued to the planned maximum sample size:

$$\text{PPoS} = P(P(p_T > p_C \mid \text{all data}) > \theta_E \mid \text{current data}).$$

This is computed by averaging over the predictive distribution of future outcomes. If PPoS falls below a threshold (e.g., 0.05), continuing the trial is unlikely to yield a positive result, making futility stopping appropriate.

Operating Characteristics

The operating characteristics of a Bayesian sequential design, including the Type I error rate, power, and expected sample size, are determined through simulation. For a given set of true parameter values (π_C, π_T) , one simulates many trials, applies the stopping rules at each interim analysis, and tabulates the outcomes. This approach allows calibration of the thresholds θ_E and θ_F to achieve desired frequentist properties if regulatory requirements demand it.

Berry emphasized that Bayesian methods do not require such calibration for logical validity, the posterior probability is always a coherent measure of evidence regardless of the stopping rule. However, demonstrating acceptable operating characteristics facilitates regulatory acceptance and provides assurance that the design performs well across plausible scenarios.

The FDA has increasingly recognized the value of Bayesian methods in drug development, issuing guidance documents on their use (U.S. Food and Drug Administration 2010). Berry Consultants has designed hundreds of adaptive trials using these methods, demonstrating their practical viability across therapeutic areas including oncology, cardiovascular disease, and rare diseases.

Example 6.10 (Bayesian Sequential Trial for a Rare Disease). Consider a Phase II trial for a rare autoimmune condition where the standard of care has

a 30% response rate. A new therapy is hypothesized to improve response to 50%. Due to the rarity of the condition, we plan for a maximum of 60 patients (30 per arm) with interim analyses every 10 patients per arm.

We use weakly informative Beta(1,1) priors for both response rates. The efficacy threshold is $\theta_E = 0.95$ for declaring $P(p_T > p_C \mid \text{data}) > 0.95$, and the futility threshold is $\theta_F = 0.10$.

```
# Simulate patient outcomes
n_control <- n_max_per_arm
n_treatment <- n_max_per_arm
control_outcomes <- rbinom(n_control, 1, p_control_true)
treatment_outcomes <- rbinom(n_treatment, 1, p_treatment_true)
```

Now we perform the sequential analysis according to the rules we have set up. We stop the trial if the posterior probability of the treatment being superior to the control is greater than 0.95 or less than 0.10.

The sequential analysis proceeds as follows. At each interim analysis, we compute the posterior distributions for both response rates using the accumulated data. For the control arm, the posterior is $\text{Beta}(\alpha_{\text{prior}} + x_C, \beta_{\text{prior}} + n - x_C)$, where x_C is the number of responders observed. Similarly, for the treatment arm, the posterior is $\text{Beta}(\alpha_{\text{prior}} + x_T, \beta_{\text{prior}} + n - x_T)$.

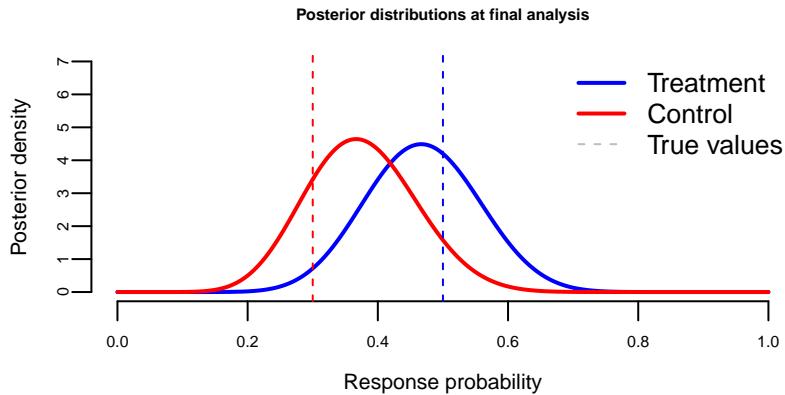
To evaluate the probability that the treatment response rate exceeds the control rate, $P(p_T > p_C \mid \text{data})$, we draw Monte Carlo samples from both posterior distributions and compute the proportion of samples where $p_T > p_C$. This probability is then compared against our decision thresholds: if it exceeds $\theta_E = 0.95$, we stop for efficacy; if it falls below $\theta_F = 0.10$, we stop for futility; otherwise, we continue enrolling patients.

The table below shows the results at each interim analysis, including the accumulated number of responders in each arm and the posterior probability of treatment superiority:

Table 6.6: Sequential analysis results

Interim	N per arm	Control successes	Treatment successes	P(Treatment > Control)	Decision
1	10	3	5	0.81	Continue
2	20	8	11	0.83	Continue
3	30	11	14	0.78	Continue

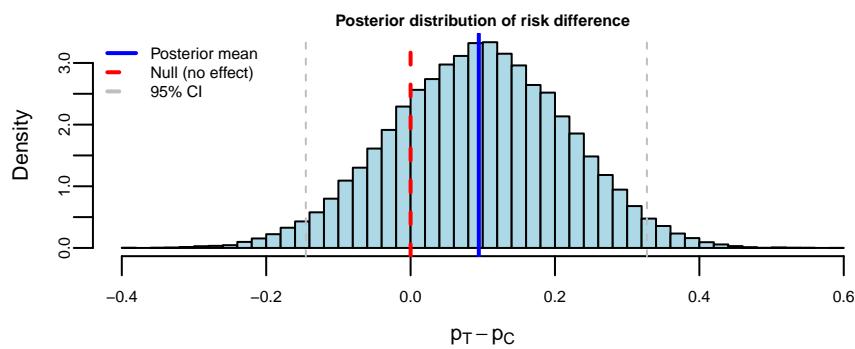
The trial data shows the accumulation of evidence across interim analyses. Let us visualize the posterior distributions at the final interim analysis:



We can also examine the posterior distribution of the treatment effect (risk difference):

```
# Posterior of risk difference via Monte Carlo
set.seed(123)
p_C_samples <- rbeta(50000, alpha_C_final, beta_C_final)
p_T_samples <- rbeta(50000, alpha_T_final, beta_T_final)
risk_diff <- p_T_samples - p_C_samples

## Posterior summary for risk difference (p_T - p_C):
## Mean: 0.095
## 95% Credible Interval: [ -0.14 , 0.33 ]
## P(p_T > p_C): 0.78
## P(p_T > p_C + 0.10): 0.49
```



Example 6.11 (Operating Characteristics via Simulation). To understand how the Bayesian sequential design performs across different scenarios, we simulate its operating characteristics. This involves running many hypothetical trials under various true parameter values and recording the outcomes. The simulation proceeds by generating a large number of hypothetical trials under each scenario. For each trial, patient outcomes are generated according to the true response probabilities, and the sequential monitoring procedure is applied at each interim analysis point. At each look, posterior distributions are updated and the probability that treatment is superior to control is computed via Monte Carlo sampling. The trial stops early for efficacy if this probability exceeds the efficacy threshold $\theta_E = 0.95$, or for futility if it falls below the futility threshold $\theta_F = 0.10$. If neither stopping criterion is met at any interim analysis, the trial continues to the maximum sample size.

The simulation tracks three key operating characteristics: the decision reached (efficacy, futility, or maximum sample size), the final sample size at which the trial stopped, and the posterior probability of treatment superiority at that point. By replicating this process across many trials under different true parameter values—ranging from no treatment effect to large effects—we can assess the design’s ability to correctly identify effective treatments, avoid false positives, and efficiently use resources by stopping early when appropriate.

The simulation code implements this procedure by defining a `simulate_trial()` function that generates patient outcomes under specified true response probabilities, then applies the sequential monitoring rules at interim points ($n = 10, 20, 30$ per arm). For each interim analysis, it draws Monte Carlo samples from the posterior distributions of p_C and p_T and computes the probability of treatment superiority. The trial stops early for efficacy if this probability exceeds $\theta_E = 0.95$, for futility if it falls below $\theta_F = 0.10$, or continues to the maximum sample size otherwise.

Four scenarios are evaluated by running 1,000 simulated trials under each: no treatment effect ($p_T = 0.30 = p_C$), small effect ($p_T = 0.40$ vs $p_C = 0.30$), moderate effect ($p_T = 0.50$), and large effect ($p_T = 0.60$). The operating characteristics—probability of declaring efficacy, probability of stopping for futility, and expected sample size per arm—are computed by aggregating outcomes across the replicated trials.

Table 6.7: Operating characteristics of the Bayesian sequential design

Scenario	True p_C	True p_T	P(Efficacy)	P(Futility)	E[N]
Null (no effect)	0.3	0.3	0.095	0.181	26.28
Small effect	0.3	0.4	0.281	0.071	25.53
Moderate effect	0.3	0.5	0.543	0.028	22.91
Large effect	0.3	0.6	0.807	0.002	19.59

The results in Table 6.7 reveal several important properties of the sequential design. Under the null hypothesis (no treatment effect), the probability of declaring efficacy is 9.5%, which represents the Type I error rate—the false positive rate when there is truly no benefit. This exceeds the traditional 5% threshold, reflecting the fact that the sequential procedure with its multiple interim looks increases the chance of a spurious efficacy declaration. If stricter Type I error control were required, the efficacy threshold θ_E could be raised above 0.95.

The power of the design—the probability of correctly identifying an effective treatment—increases substantially as the true treatment effect grows: 28.1% for the small effect, 54.3% for moderate, and 80.7% for large effects. The relatively modest power for small effects (10 percentage point difference in response rates) reflects the inherent difficulty of detecting subtle improvements with limited sample sizes. More patients would be needed to reliably identify small but clinically meaningful benefits.

Perhaps most striking is the expected sample size, which demonstrates the efficiency gains from sequential stopping. Under the null hypothesis, the average trial uses only 26.3 patients per arm rather than the maximum of 30, as 18.1% of trials stop early for futility. When the treatment effect is large, the average sample size drops to just 19.6 per arm, with 80.7% of trials stopping early for efficacy. This adaptive efficiency means fewer patients are exposed to inferior treatments and trial results become available sooner, accelerating the translation of effective therapies into clinical practice.

6.5.4 Sequential Analysis for Rare Diseases

Sequential and adaptive designs are particularly valuable for clinical trials in rare diseases, where small patient populations make traditional fixed-sample designs impractical. With potentially only hundreds or even dozens of patients worldwide having a particular condition, every enrolled patient provides precious information that must be used efficiently.

Several innovative approaches have emerged for rare disease trials. The small n Sequential Multiple Assignment Randomized Trial (snSMART) design (B. Wei et al. 2018) provides a framework for testing multiple treatments within a single trial, re-randomizing non-responding patients to alternative therapies in subsequent stages. This design achieves increased statistical power over traditional single-stage designs by leveraging information from all treatment sequences.

Hilgers, Roes, and Stallard (2016) provided an overview of design options for achieving valid randomized clinical trials in rare diseases, emphasizing that sequential procedures can substantially reduce the expected sample size while maintaining statistical validity. Their work demonstrated that group

sequential designs and adaptive approaches can cut average trial sizes by 30–50% compared to fixed designs.

The application of Bayesian methods specifically to rare disease trials has been systematically reviewed by Cong Chen et al. (2022), who identified both the opportunities and hurdles in this setting. The Bayesian framework naturally accommodates informative priors based on historical data, external controls, or expert elicitation, which is particularly valuable when patient populations are too small for purely data-driven inference. However, the authors caution that prior sensitivity analyses are essential when sample sizes are small, as the prior can dominate posterior conclusions.

Benda et al. (2016) evaluated the performance of sequential methods specifically in small-sample settings with normally distributed responses. They found that careful attention to the distribution of test statistics is necessary to maintain nominal Type I error rates, as large-sample approximations may not hold. Their recommendations provide practical guidance for implementing sequential designs in trials with limited participants.

6.5.5 Multi-Armed Bandit Experiments

A closely related approach to sequential analysis is the multi-armed bandit, a framework that has changed how online experiments are conducted in the digital economy. While classical A/B tests require predetermining sample sizes and waiting for experiments to conclude before making decisions, multi-armed bandits adaptively allocate traffic based on accumulating evidence. This approach, implemented in platforms like Google Analytics Content Experiments (Steven L. Scott 2013; Steven L. Scott 2015), represents a practical application of Bayesian sequential analysis at a massive scale.

A multi-armed bandit is a type of experiment characterized by two key features: the goal is to find the best or most profitable action (or *arm*), and the randomization distribution can be updated as the experiment progresses. The colorful name originates from a hypothetical scenario where a gambler faces several slot machines (colloquially called one-armed bandits) with potentially different, unknown payoff rates. The gambler wants to find the machine with the best payout, but also wants to maximize winnings during the search. This creates the fundamental tension between *exploiting* arms that have performed well in the past and *exploring* new or seemingly inferior arms that might perform even better.

The theoretical resolution to this trade-off was provided by John Gittins in 1979. He proved that for the discounted multi-armed bandit problem, optimal policies have a simple index form. The *Gittins index* is a value assigned to each arm that depends only on its own state (its current posterior distribution) and the discount factor. The optimal strategy is to always play the arm with

the highest index. Intuitively, the index represents the guaranteed reward rate that would make a rational agent indifferent between the arm and the guaranteed reward. While the Gittins index provides the optimal solution, it can be computationally intensive to calculate. In practice, especially for large-scale internet applications, simpler heuristics like Thompson sampling (discussed below) are widely used and often achieve near-optimal performance.

The Bayesian Approach to Bandits

The Bayesian framework provides an elegant solution to the bandit problem. Suppose we have K arms (variations), each with an unknown success probability θ_k for $k = 1, \dots, K$. Using Bayes' theorem, we can compute the probability that each arm is the best based on observed data. Let x_k denote the number of successes and n_k the number of trials for arm k . With a Beta prior $\theta_k \sim \text{Beta}(\alpha, \beta)$, the posterior after observing data is

$$\theta_k | x_k, n_k \sim \text{Beta}(\alpha + x_k, \beta + n_k - x_k).$$

The probability that arm k is optimal is

$$P(\theta_k > \theta_j \text{ for all } j \neq k | \text{data}),$$

which can be computed via Monte Carlo integration by sampling from the posterior distributions and counting how often arm k produces the largest sample.

The key insight from Steven L. Scott (2015) is that these posterior probabilities of optimality can serve directly as allocation weights. An arm that appears to be doing well receives more traffic, while an arm that is clearly underperforming receives less. The adjustments consider sample size and performance metrics together, providing confidence that we are adjusting for real performance differences rather than random chance.

Thompson Sampling

The algorithm that implements this Bayesian allocation is called Thompson sampling (Thompson 1933), one of the oldest heuristics for the bandit problem, dating to 1933. For binary outcomes with Beta priors, the algorithm proceeds as follows:

1. Initialize $\alpha_k = \beta_k = 1$ for each arm (uniform prior).
2. At each decision point, sample $\tilde{\theta}_k$ from the current posterior $\text{Beta}(\alpha_k, \beta_k)$ for each arm.
3. Select the arm with the highest sampled value: $a_t = \arg \max_k \tilde{\theta}_k$.

4. Observe the outcome $Y_t \in \{0, 1\}$ and update: $\alpha_{a_t} \leftarrow \alpha_{a_t} + Y_t$, $\beta_{a_t} \leftarrow \beta_{a_t} + (1 - Y_t)$.

Thompson sampling automatically balances exploration and exploitation. Arms with high posterior means are selected frequently (exploitation), but arms with high posterior variance, which indicates uncertainty, also have a chance of producing high samples (exploration). As data accumulates, posteriors concentrate around true values, and the algorithm increasingly exploits the best arm.

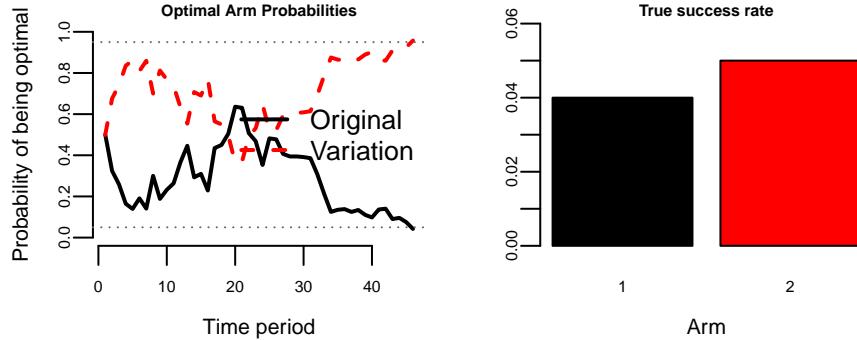
Benefits Over Classical A/B Testing

Experiments based on multi-armed bandits are typically much more efficient than classical A/B experiments based on hypothesis testing. As Steven L. Scott (2013) explains, they are just as statistically valid but can produce answers far more quickly. The efficiency gains arise from two sources: traffic moves toward winning variations gradually rather than waiting for a final answer, and samples that would have gone to obviously inferior variations can be assigned to potential winners.

Consider a concrete example from Steven L. Scott (2013). Suppose a website has a 4% conversion rate, and a new variation actually converts at 5%. A standard power calculation for detecting this difference at 95% confidence requires 22,330 observations (11,165 per arm). At 100 visits per day, this experiment would take 223 days to complete. In a classical experiment, you wait 223 days, run the hypothesis test, and get your answer.

With a multi-armed bandit, the experiment can finish much sooner. In simulations, the bandit found the correct arm in 96.4% of cases (about the same error rate as the classical test), but the average experiment duration was only 66 days, saving 157 days of testing. The savings compound when experiments have more arms, because the classical approach requires Bonferroni-type corrections for multiple comparisons while the bandit naturally handles multiple arms through the posterior probability framework.

Example 6.12 (Reproducing Scott's Bandit Experiment). We reproduce the simulation from Steven L. Scott (2013). The setup involves an original page with 4% conversion rate and a variation with 5% conversion rate, with 100 visits per day. A classical power calculation requires 223 days; we will see how the bandit performs.



```
## Experiment ended on day: 46
## Days saved vs classical: 177
```

The left panel shows the probability that each arm is optimal over time. The two curves are complementary (summing to 1) and fluctuate until eventually one crosses the 95% threshold. The right panel shows the true success rates that are unknown to the algorithm.

To understand the distribution of outcomes, we run 500 simulations and compare to the classical experiment:

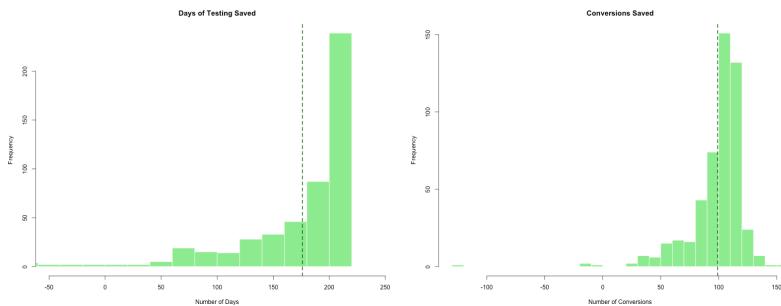


Figure 6.4: Distribution of days saved and conversions saved

The histograms in Figure 6.4 show the distribution of days saved (left) and conversions saved (right) compared to the classical 223-day experiment. The key results from 500 simulations are summarized in Table 6.8:

Table 6.8: Simulation results for multi-armed bandit vs. classical experiment ($n = 500$)

Metric	Value
Correct arm identified	469 / 500 (93.8%)
Average days to finish	46.9
Average days saved	176.1
Average conversions saved	99.1

On average, the bandit saves approximately 176 days of testing (about 79% reduction) while achieving the same statistical validity (finding the correct arm 93.8% of the time, comparable to the 95% confidence level of the classical test). The conversion savings come from two sources: ending the experiment earlier and allocating more traffic to the better-performing variation during the experiment.

Stopping Rules, Value Remaining, and Regret

Several questions arise in practice: when should we stop the experiment? How do we quantify the potential benefit of continuing? Steven L. Scott (2015) describes two complementary stopping criteria. The first is the probability that each variation beats the original; if we are 95% confident that a variation is best, a winner can be declared. The second criterion is the *value remaining in the experiment*, which measures the expected improvement from switching away from the current champion. When there is at least a 95% probability that the value remaining is less than 1% of the champion's conversion rate, the experiment can end.

Another perspective is *regret*, the cumulative cost of not always selecting the optimal arm. These concepts, along with detailed mathematical treatment and code examples for computing stopping criteria, are covered in Chapter 9 (see the section on Multi-Armed Bandits, specifically “When to End Experiments”).

This framework also extends to clinical trials, where Villar, Bowden, and Watson (2015) reviewed how response-adaptive randomization can reduce the number of patients receiving inferior treatments. For contextual bandits, design considerations, and extensions to reinforcement learning, see Chapter 9.

6.5.6 Practical Considerations

Sequential analysis requires careful consideration of several practical issues. First, the definition of *sufficient evidence* must be specified in advance,

whether through posterior probability thresholds, Bayes factor bounds, or expected utility calculations. Second, the frequency of interim analyses affects operational aspects of trials, including regulatory interactions and data monitoring committee responsibilities. Third, the potential for early stopping must be balanced against the need for long-term safety data and secondary endpoint analyses.

The choice between Bayesian and frequentist sequential methods often depends on regulatory requirements and institutional preferences. However, the coherence of Bayesian inference under optional stopping, the natural incorporation of prior information, and the interpretability of posterior probabilities make Bayesian sequential analysis an increasingly attractive option, particularly in challenging settings like rare disease trials where traditional approaches are impractical.

6.6 Examples and Paradoxes

This section provides a number of paradoxes arising when using different hypothesis testing procedures. The common strands of the examples will be discussed at the end of the section.

Example 6.13 (Neyman-Pearson tests). Consider testing $H_0 : \mu = \mu_0$ versus $H_1 : \mu = \mu_1$, $y_t \sim \mathcal{N}(\mu, \sigma^2)$ and $\mu_1 > \mu_0$. For this simple test, the likelihood ratio is given by

$$\mathcal{LR}_{0,1} = \frac{\exp\left(-\frac{1}{2\sigma^2}\sum_{t=1}^T (y_t - \mu_0)^2\right)}{\exp\left(-\frac{1}{2\sigma^2}\sum_{t=1}^T (y_t - \mu_1)^2\right)} = \exp\left(-\frac{T}{\sigma^2}(\mu_1 - \mu_0)\left(\bar{y} - \frac{1}{2}(\mu_0 + \mu_1)\right)\right).$$

Since $\text{BF}_{0,1} = \mathcal{LR}_{0,1}$, assuming equal prior probabilities and symmetric losses, the Bayesian accepts H_0 if $\text{BF}_{0,1} > 1$. Thus, the Bayes procedure rejects H_0 if $\bar{y} > \frac{1}{2}(\mu_0 + \mu_1)$ for any T and σ^2 , with μ_0, μ_1, T , and σ^2 determining the strength of the rejection. If $\text{BF}_{0,1} = 1$, there is equal evidence for the two hypotheses.

The NP procedure proceeds by first setting $\alpha = 0.05$, and rejects when $\mathcal{LR}_{0,1}$ is large. This is equivalent to rejecting when \bar{y} is large, generating an ‘optimal’ rejection region of the form $\bar{y} > c$. The cutoff value c is calibrated via the size of the test,

$$P[\text{reject } H_0 | H_0] = P[\bar{y} > c | \mu_0] = P\left[\frac{(\bar{y} - \mu_0)}{\sigma/\sqrt{T}} > \frac{(c - \mu_0)}{\sigma/\sqrt{T}} | H_0\right].$$

The size equals α if $\sqrt{T}(c - \mu_0)/\sigma = z_\alpha$. Thus, the NP test rejects if then if $\bar{y} > \mu_0 + \sigma z_\alpha/\sqrt{T}$. Notice that the test rejects regardless of the value of μ_1 ,

which is rather odd, since μ_1 does not enter into the size of the test only the power. The probability of a type II error is

$$\beta = P[\text{accept } H_0 \mid H_1] = P\left[\bar{y} \leq \mu_0 + \frac{\sigma}{\sqrt{T}} z_\alpha \mid H_1\right] = \int_{-\infty}^{\mu_0 + \frac{\sigma}{\sqrt{T}} z_\alpha} p(\bar{y} \mid \mu_1) d\bar{y},$$

where $p(\bar{y} \mid \mu_1) \sim \mathcal{N}(\mu_1, \sigma^2/T)$.

These tests can generate strikingly different conclusions. Consider a test of $H_0 : \mu = 0$ versus $H_1 : \mu = 5$, based on $T = 100$ observations drawn from $y_t \sim \mathcal{N}(\mu, 10^2)$ with $\bar{y} = 2$. For NP, since $\sigma/\sqrt{T} = 1$, \bar{y} is two standard errors away from 0, thus H_0 is rejected at the 5% level (the same conclusion holds for p -values). Since $p(\bar{y} = 2 \mid H_0) = 0.054$ and $p(\bar{y} = 2 \mid H_1) = 0.0044$, the Bayes factor is $\text{BF}_{0,1} = 12.18$ and $P(H_0 \mid y) = 92.41\%$. Thus, the Bayesian is quite sure the null is true, while Neyman-Pearson reject the null.

The paradox can be seen in two different ways. First, although \bar{y} is actually closer to μ_0 than μ_1 , the NP test rejects H_0 . This is counterintuitive and makes little sense. The problem is one of calibration. The classical approach develops a test such that 5% of the time, a correct null would be rejected. The power of the test is easy to compute and implies that $\beta = 0.0012$. Thus, this testing procedure will virtually never accept the null if the alternative is correct. For Bayesian procedure, assuming the prior odds is 1 and $L_0 = L_1$, then $\alpha = \beta = 0.0062$. Notice that the overall probability of making an error is 1.24% in the Bayesian procedure compared to 5.12% in the classical procedure. It should seem clear that the Bayesian approach is more reasonably, absent a specific motivation for inflating α . Second, suppose the null and alternative were reversed, testing $H_0 : \mu = \mu_1$ versus $H_1 : \mu = \mu_0$. In the previous example, the Bayes approach gives the same answer, while NP once again rejects the null hypothesis! Again, this result is counterintuitive and nonsensical, but is common when arbitrarily fixing α , which essentially hardwires the test to over-reject the null.

Example 6.14 (Lindley's paradox). Consider the case of testing whether or not a coin is fair, based on observed coin flips,

$$H_0 : \theta = \frac{1}{2} \text{ versus } H_1 : \theta \neq \frac{1}{2},$$

based on T observations from $y_t \sim \text{Ber}(\theta)$. As an example, Table 6.9 provides 4 datasets of differing lengths. Prior to considering the formal hypothesis tests, form your own opinion on the strength of evidence regarding the hypothesis in each data set. It is common for individuals, when confronted with this data to conclude that the fourth sample provides the strongest of evidence for the null and the first sample the weakest.

Table 6.9: Lindley's paradox

	#1	#2	#3	#4
# Flips	50	100	400	10,000
# Heads	32	60	220	5098
Percentage of heads	64	60	55	50.98

Fisher's solution to the problem posits an unbiased estimator, the sample mean, and computes the t -statistic, which is calculated under H_0 :

$$t(y) = \frac{\bar{y} - E[\bar{y} | \theta_0]}{se(\bar{y})} = \sqrt{T} (2\hat{\theta} - 1),$$

where $se(\bar{y})$ is the standard error of \bar{y} . The Bayesian solution requires marginal likelihood under the null and alternative, which are

$$p(y | \theta_0 = 1/2) = \prod_{t=1}^T p(y_t | \theta_0) = \left(\frac{1}{2}\right)^{\sum_{t=1}^T y_t} \left(\frac{1}{2}\right)^{T - \sum_{t=1}^T y_t} = \left(\frac{1}{2}\right)^T, \quad (6.1)$$

and, from Equation 6.1, $p(y | H_1) = B(\alpha_T, \beta_T) / B(\alpha, \beta)$ assuming a beta prior distribution.

To compare the results, note first that in the datasets given above, $\hat{\theta}$ and T generate $t_\alpha = 1.96$ in each case. Thus, for a significance level of $\alpha = 5\%$, the null is rejected for each sample size. Assuming a flat prior distribution, the Bayes factors are

$$BF_{0,1} = \begin{cases} 0.8178 & \text{for } N = 50 \\ 1.0952 & \text{for } N = 100 \\ 2.1673 & \text{for } N = 400 \\ 11.689 & \text{for } N = 10000 \end{cases},$$

showing increasingly strong evidence in favor of H_0 . Assuming equal prior weight for the hypotheses, the posterior probabilities are 0.45, 0.523, 0.684, and 0.921, respectively. For the smallest samples, the Bayes factor implies roughly equal odds of the null and alternative. As the sample size increase, the weight of evidence favors the null, with a 92% probability for $N = 10K$.

Next, consider testing $H_0 : \theta_0 = 0$ vs. $H_1 : \theta_0 \neq 0$, based on T observations from $y_t \sim \mathcal{N}(\theta_0, \sigma^2)$, where σ^2 is known. This is the formal example used by Lindley to generate his paradox. Using p -values, the hypothesis is rejected if the t -statistic is greater than t_α . To generate the paradox, consider datasets that are exactly t_α standard errors away from \bar{y} , that is, $\bar{y}^* = \theta_0 + \sigma t_\alpha / \sqrt{n}$, and a uniform prior over the interval $(\theta_0 - I/2, \theta_0 + I/2)$. If p_0 is the probability

of the null, then,

$$\begin{aligned}
 P(\theta = \theta_0 \mid \bar{y}^*) &= \frac{\exp\left(-\frac{1}{2}\frac{T}{2}\right) p_0}{\exp\left(-\frac{1}{2}\frac{T(\bar{y}^* - \theta_0)^2}{\sigma^2}\right) p_0 + (1 - p_0)} \\
 &= \frac{\exp\left(-\frac{1}{2}t_\alpha^2\right) p_0}{\exp\left(-\frac{1}{2}t_\alpha^2\right) p_0 + \frac{(1-p_0)}{I} \int_{\theta_0-I/2}^{\theta_0+I/2} \exp\left(-\frac{1}{2}\left(\frac{(\bar{y}^* - \theta)}{\sigma/\sqrt{T}}\right)^2\right) d\theta} \\
 &\geq \frac{\exp\left(-\frac{1}{2}t_\alpha^2\right) p_0}{\exp\left(-\frac{1}{2}t_\alpha^2\right) p_0 + \frac{(1-p_0)}{I} \sqrt{2\pi\sigma^2/T}} \rightarrow 1 \text{ as } T \rightarrow \infty.
 \end{aligned}$$

i ^{2}/T\} \rightarrow 1 \text{ as } T \rightarrow \infty. \% \end{aligned} \} In large samples, the posterior probability of the null approaches 1, whereas Fisher always reject the null. It is important to note that this holds for any t_α , thus even if the test were performed at the 1% level or lower, the posterior probability would eventually reject the null.

6.7 Prior Sensitivity

The paradoxes discussed above, particularly Lindley's Paradox, often invite criticism regarding the choice of the prior distribution. A common frequentist counter-argument is that the Bayesian result is driven by a specific, perhaps "biased", prior. How do we know that the chosen prior is not artificially inflating the probability of the null? One elegant way to address this is to search over classes of priors to find the one that *minimizes* the evidence for the null hypothesis, effectively constructing a "worst-case" Bayesian scenario.

To see this, consider the case of testing $H_0 : \mu_0 = 0$ vs. $H_1 : \mu_0 \neq 0$ with observations drawn from $y_t \sim \mathcal{N}(\theta_0, \sigma^2)$, with σ known. With equal prior null and alternative probability, the probability of the null is $p(H_0 \mid y) = (1 + (\text{BF}_{0,1})^{-1})^{-1}$. Under the null,

$$p(y \mid H_0) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{T}{2}} \exp\left(-\frac{1}{2}\left(\frac{(\bar{y} - \theta_0)}{\sigma/\sqrt{T}}\right)^2\right).$$

The criticism applies to the priors under the alternative. To analyze the sensitivity, consider four classes of priors under the alternative: (a) the class of normal priors, $p(\theta \mid H_1) \sim \mathcal{N}(a, A)$; (b) the class of all symmetric unimodal prior distributions; (c) the class of all symmetric prior distributions; and (d)

the class of all proper prior distributions. These classes provide varying degrees of prior information, allowing a thorough examination of the strength of evidence.

In the first case, consider the standard conjugate prior distribution, $p(\mu | H_1) \sim \mathcal{N}(\mu_0, A)$. Under the alternative,

$$\begin{aligned} p(y | H_1) &= \int p(y | \mu, H_1) p(\mu | H_1) d\mu \\ &= \int p(\bar{y} | \mu, H_1) p(\mu | H_1) d\mu, \end{aligned}$$

using the fact that \bar{y} is a sufficient statistic. Noting that $p(\bar{y} | \mu, H_1) \sim N(\mu, \sigma^2/T)$ and $p(\mu | H_1) \sim N(\mu_0, A)$, we can use the “substitute” instead of integrate trick to assert that

$$\bar{y} = \mu_0 + \sqrt{A}\eta + \sqrt{\sigma^2/T}\varepsilon,$$

where η and ε are standard normal. Then, $p(\bar{y} | H_1) \sim \mathcal{N}(\mu_0, A + \sigma^2/T)$. Thus,

$$\text{BF}_{0,1} = \frac{p(y | H_0)}{p(y | H_1)} = \frac{p(\bar{y} | H_0)}{p(\bar{y} | H_1)} = \frac{(\sigma^2/T)^{-\frac{1}{2}}}{(\sigma^2/T + A)^{-\frac{1}{2}}} \frac{\exp(-\frac{1}{2}t^2)}{\exp(-\frac{1}{2}\frac{z^2\sigma^2/T}{A+\sigma^2/T})}.$$

To operationalize the test, A must be selected. A is chosen to minimizing the posterior probabilities of the null, with $P_{norm}(H_0 | y)$ being the resulting lower bound on the posterior probability of the null. For $z \geq 1$, the lower bound on the posterior probability of the null is

$$P_{norm}(H_0 | y) = [1 + \sqrt{e} \exp(-.5t^2)]^{-1},$$

which is derived in a reference cited in the notes. This choice provides a maximal bias of the Bayesian approach toward rejecting the null. It is important to note that this is not a reasonable prior, as it was intentionally constructed to bias the null toward rejection.

For the class of all proper prior distributions, it is also easy to derive the bound. From equation above, minimizing the posterior probability is equivalent to minimizing the Bayes factor,

$$\text{BF}_{0,1} = \frac{p(y | H_0)}{p(y | H_1)}.$$

Since

$$p(y | H_1) = \int p(y | \theta, H_1) p(\theta | H_1) d\theta \leq p(y | \hat{\theta}_{MLE}, H_1),$$

where $\hat{\theta}_{MLE} = \arg \max_{\theta \neq 0} p(y | \theta)$. The maximum likelihood estimator, maximizes the probability of the alternative, and provides a lower bound on the Bayes factor,

$$\underline{BF}_{0,1} = \frac{p(y | H_0)}{\sup_{\theta \neq 0} p(y | \theta)}.$$

In this case, the bound is particularly easy to calculate and is given by

$$P_{all}(H_0 | y) = \left(1 + \exp\left(-\frac{t^2}{2}\right)\right)^{-1}.$$

A reference cited in the notes provides the bounds for the second and third cases, generating $P_{s,u}(H_0 | y)$ and $P_s(H_0 | y)$, respectively. All of the bounds only depend on the t -statistic and constants.

Table 6.10 reports the t -statistics and associated p -values, with the remaining columns provide the posterior probability bounds. For the normal prior and choosing the prior parameter A to minimize the probability of the null, the posterior probability of the null is much larger than the p -value, in every case. For the standard case of a t -statistic of 1.96, $P(H_0 | y)$ is more than six times greater than the p -value. For $t = 2.576$, $P(H_0 | y)$ is almost 13 times greater than the p -value. These probabilities fall slightly for more general priors. For example, for the class of all priors, a t-statistic of 1.96/2.576 generates a lower bound for the posterior probability of 0.128/0.035, more than 2/3 times the p -value.

Table 6.10: Comparison of strength of evidence against the point null hypothesis. The numbers are reproduced from Berger (1986).

t -stat	p -value	$P_{norm}(H_0 y)$	$P_{s,u}(H_0 y)$	$P_s(H_0 y)$	$P_{all}(H_0 y)$
1.645	0.100	0.412	0.39	0.34	0.205
1.960	0.050	0.321	0.29	0.227	0.128
2.576	0.010	0.133	0.11	0.068	0.035
3.291	0.001	0.0235	0.018	0.0088	0.0044

6.8 The difference between p-values and Bayesian evidence

Suppose that you routinely reject two-sided hypotheses at a fixed level of significance, $\alpha = 0.05$. Furthermore, suppose that half the experiments under the null are actually true: $p(H_0) = p(H_1) = \frac{1}{2}$.

The observed p-value is not a probability in any real sense. The observed t-value is a realization of a statistic that happens to be $N(0, 1)$ under the null hypothesis. Suppose that we observe $t = 1.96$.

Then the *maximal evidence* against the null hypothesis, which corresponds to $t = 0$, will be achieved by evaluating the likelihood ratio at the observed t-ratio. We get

$$\frac{p(y | H_0)}{p(y | H_1)} \geq \frac{p(y | \theta = \theta_0)}{p(y | \theta = \hat{\theta})}$$

Technically, $p(y | H_1) = \int p(y | \theta)p(\theta | H_1)d\theta \leq p(y | \hat{\theta})$.

For testing, this gives:

$$\frac{p(y | H_0)}{p(y | H_1)} \geq \frac{\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\cdot 1.96^2}}{\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\cdot 0^2}} = 0.146$$

In terms of probabilities, with $p(H_0) = p(H_1)$, we have:

$$p(H_0 | y) = \frac{1}{1 + \frac{p(y | H_1)}{p(y | H_0)}} \geq 0.128$$

Hence, there's still a 12.8% chance that the null is true! That's very different from the p-value of 5%.

Moreover, among experiments with p-values of 0.05, at least 28.8% will actually turn out to be true nulls (Sellke, Bayarri, and Berger 2001)! Put another way, the probability of rejecting a true null *conditional* on the observed $p = 0.05$ is at least 30%. You are throwing away good null hypotheses and claiming you have found effects!

6.9 Jeffreys' Decision Rule

Jeffreys (1998) provided a famous rule for hypothesis testing. Consider testing $H_0 : \beta = 0$ versus $H_1 : \beta \neq 0$ with a t-statistic $\frac{\hat{\beta}}{s_{\hat{\beta}}}$. Jeffreys proposed the rule:

$$\frac{\hat{\beta}}{s_{\hat{\beta}}} > \log\left(\frac{2n}{\pi}\right)$$

This result can be understood through the **Dickey-Savage density ratio**, which states that for a sharp null hypothesis $H_0 : \beta = 0$ nested within an

alternative H_1 , the Bayes Factor is the ratio of the posterior density to the prior density evaluated at the null value:

$$BF_{01} = \frac{p(\beta = 0 \mid y, H_1)}{p(\beta = 0 \mid H_1)}.$$

Assuming a Cauchy(0, σ) prior for β under H_1 , the prior density at the null is $p(0 \mid H_1) = 1/(\pi\sigma)$. For the posterior, we can approximate it using a normal distribution centered at the MLE $\hat{\beta}$ with standard error $s_{\hat{\beta}}$. Evaluating this approximate posterior at 0 gives:

$$p(0 \mid y, H_1) \approx \frac{1}{\sqrt{2\pi}s_{\hat{\beta}}} \exp\left(-\frac{1}{2}\left(\frac{0-\hat{\beta}}{s_{\hat{\beta}}}\right)^2\right).$$

Substituting these into the ratio, the condition for evidence neutrality ($BF = 1$) becomes:

$$\frac{\frac{1}{\sqrt{2\pi}s_{\hat{\beta}}}e^{-t^2/2}}{1/(\pi\sigma)} = 1 \implies \sqrt{\frac{n\pi}{2}}e^{-t^2/2} \approx 1,$$

where we've used the approximation $s_{\hat{\beta}} \approx \sigma/\sqrt{n}$. Jeffreys' rule $\hat{\beta}/s_{\hat{\beta}} > \log(2n/\pi)$ provides a heuristic approximation to this boundary.

These critical values differ substantially from the fixed frequentist thresholds:

Table 6.11: Jeffreys' decision rule critical values Jeffreys (1998) (p. 379)

n	$\hat{\beta}/\sigma_{\hat{\beta}}$
5	1.16
10	1.85
100	4.15
100,000	11.06

For instance, when $n = 10$, we have $\hat{\beta}/\sigma_{\hat{\beta}} = \log(20/\pi) = 1.85$, and for $n = 100$, we have $\hat{\beta}/\sigma_{\hat{\beta}} = \log(200/\pi) = 4.15$.

Jeffreys then explains the consequences of this sample-size dependence: traditional fixed critical values like 1.96 do not properly account for the evidence provided by larger sample sizes.

6.10 Cromwell's Rule

The discussion of hypothesis testing throughout this chapter reveals a fundamental tension between the desire for certainty and the reality of uncertainty in statistical inference. This tension is captured by **Cromwell's Rule**, a principle that serves as a philosophical foundation for Bayesian hypothesis testing.

We can write Bayes rule for updating models as follows:

$$p(M | D) = \frac{p(D | M)}{p(D)} p(M)$$

Thus, if $p(M) = 0$, then $p(M | D) = 0$ for all D .

This mathematical result has profound implications: if you assign zero prior probability to a hypothesis, no amount of evidence can ever change your mind. The posterior probability remains zero regardless of how strongly the data might support that hypothesis.

This principle is named after Oliver Cromwell's famous plea to the Church of Scotland in 1650:

I beseech you, in the bowels of Christ, think it possible you may be mistaken

Cromwell's appeal for intellectual humility resonates deeply with the Bayesian approach to hypothesis testing. The rule suggests that we should never assign zero probability to hypotheses that could plausibly be true, as doing so makes us unable to learn from any amount of contradictory evidence.

The rule emphasizes the importance of careful prior specification. As we saw in the section on prior sensitivity, even when we try to bias our priors against the null hypothesis, the resulting posterior probabilities often remain substantially higher than corresponding p-values. Cromwell's Rule reminds us that assigning zero probability to any reasonable hypothesis is not just mathematically problematic—it's epistemologically unsound.

Cromwell's Rule further aligns with the likelihood principle discussed earlier. Just as the likelihood principle states that all relevant experimental information is contained in the likelihood function, Cromwell's Rule ensures that we remain open to learning from all possible evidence. By avoiding zero prior probabilities, we maintain the ability to update our beliefs based on observed data.

This principle serves as a philosophical foundation that unifies the various approaches to hypothesis testing discussed in this chapter, emphasizing the importance of intellectual humility and the willingness to learn from evidence in statistical inference.

7

Stochastic Processes

Yet another fundamental concept that is useful for probabilistic reasoning is a stochastic process. An instance of a process is a function $X: \Omega \rightarrow S$ from an index set Ω to a set of possible values S , called the state space. The state space of a stochastic process is the set of all possible states that the process can be in. Each state in the state space represents a possible outcome or condition of the system being modeled. The process then is the *distribution over the space of functions* from Ω to S . The term process is used because the function X is often thought of as a time-varying quantity, and the index set Ω is often interpreted as time. While time is the most common index, the index set Ω is general; for example, in geostatistics, the process is indexed by spatial coordinates rather than time. Both the index set and the state space can be discrete or continuous.

A key concept in many stochastic processes is the *Markov property*, which states that the future state of the process depends only on the current state, not on the sequence of events that preceded it. Formally, $P(X_{t+1}|X_t, X_{t-1}, \dots) = P(X_{t+1}|X_t)$.

A stochastic process is a family of random variables that describes the evolution through time of some (physical) process. We denote this by $X = \{X_t, t \in T\}$, with t representing time and $X_t = \omega$ is the state of the process at time t . We will get a realization (a.k.a. sample path). In the case when time is discrete, the realization is a sequence of observed $X = \Omega = \{\omega_1, \omega_2, \dots\}$. Common discrete time processes are Markov chains. Brownian motion is a central process in continuous time and state with almost surely continuous but nowhere differentiable paths. Poisson processes are commonly used to account for jumps in the process.

Here are some widely used stochastic processes:

1. Random Walk: A simple example where the next state depends on the current state and some random movement. In finance, stock prices are often modeled as a type of random walk.
2. Markov Chains: A process where the next state depends only on the current state and not on the path taken to reach the current state.

3. Poisson Process: Used to model the number of times an event occurs in a fixed interval of time or space, where events occur with a known constant mean rate and independently of the time since the last event.
4. Queuing Theory: Models used in operations research where the stochastic process represents the number of customers in a queue, varying over time as customers arrive and are served.
5. Brownian Motion: This process models the random movement of particles suspended in a fluid. It has applications in physics, finance (to model stock market prices), and biology.
6. Gaussian Processes: These are a collection of random variables, any finite number of which have a joint Gaussian distribution. They are used in machine learning for regression and classification tasks.

Data variability can be modeled as i.i.d. processes or as stochastic processes with serial, spatial, or other dependencies. Modeling these dependencies moves beyond descriptive statistics to predictive frameworks that quantify uncertainty. The early sections of Davison (2003) offer a primer on developing and applying stochastic models.

7.1 Brownian Motion

Brownian Motion, named after botanist Robert Brown who observed pollen particles following irregular random trajectories in water, is fundamental to stochastic processes. A one-dimensional Brownian Motion (Wiener process) B_t , $t \geq 0$ has the following properties:

- $B_0 = 0$ almost surely
- B_t has stationary independent increments: $B_t - B_s \sim N(0, t-s)$ for $0 \leq s < t$
- B_t is a continuous function of t
- For each time $t > 0$, the random variable B_t is normally distributed with mean 0 and variance t , i.e., $B_t \sim N(0, t)$.

Formally, Brownian motion is a continuous-time stochastic process B_t for $t \geq 0$ used to model random continuous evolution.

Figure 7.1 below shows three sample paths of Brownian Motion.

```
# Brownian Motion
set.seed(92)
t <- seq(0, 1, 0.001)
```

```

plot(t, cumsum(rnorm(1001, 0, sqrt(0.001))),
      type = "l", xlab = "t",
      ylab = "B_t", lwd = 2, ylim = c(-1.2, 2)
)
lines(t, cumsum(rnorm(1001, 0, sqrt(0.001))), lwd = 2, col = 2)
lines(t, cumsum(rnorm(1001, 0, sqrt(0.001))), lwd = 2, col = 3)

```

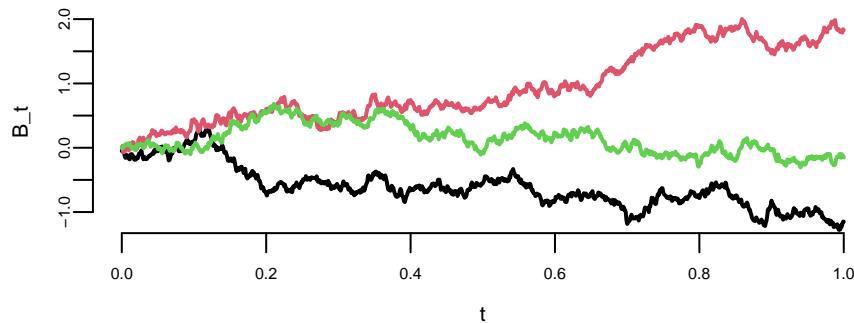


Figure 7.1: Brownian Motion

Thus, for any times $0 \leq t_1 < t_2 < \dots < t_n$, the random variables $B(t_2) - B(t_1)$, $B(t_3) - B(t_2)$, ..., $B(t_n) - B(t_{n-1})$ are independent and the function $t \mapsto B_t$ is continuous almost surely.

Some properties of Brownian Motion are:

- Scale Invariance: If B_t is a Brownian motion, then for any $a > 0$, the process $aB(t/a^2)$ is also a Brownian motion.
- Fractal Nature: Brownian motion paths are nowhere differentiable but continuous everywhere, reflecting a fractal-like nature.

The normal assumption of asset returns was first proposed in 1900 in the PhD thesis of Louis Bachelier, who was a student of Henri Poincaré. Bachelier was interested in developing statistical tools for pricing options (predicting asset returns) on the Paris stock exchange. Although Bachelier's work laid the foundation for the modern theory of stochastic processes, he was never given credit by his contemporaries, including Einstein, Levy and Borel.

In 1905 Einstein published a paper which used the same statistical model as Bachelier to describe the 1827 discovery by botanist Robert Brown, who observed that pollen particles suspended in water followed irregular random trajectories. Thus, we call the stochastic process that describes these phenomena Brownian motion. Einstein's advisor at the University of Zurich was Hermann

Minkowski who was a friend and collaborator of Poincare. Thus, it is likely Einstein knew about the work of Bachelier, but he never mentioned it in his paper. Einstein's omission of references was not without precedent; his 1905 relativity paper similarly lacked citations, famously omitting Poincaré's 1898 contributions. Poincare published a paper Poincaré (1898) on relativity theory in 1898, seven years before Einstein. This paper was published in a philosophy journal and thus Poincare avoided using any mathematical formulas except for the famous $E = mc^2$. Poincare discussed his results on relativity theory with Minkowski. Minkowski asked Einstein to read Poincare's work Arnol'd (2006). However, Einstein never referenced the work of Poincare until 1945. One of the reviewers for the 1905 paper on relativity by Einstein was Poincare and he wrote a very positive review mentioning it as a breakthrough. When Minkowski asked Poincare why he did not claim his priority on the theory, Poincare replied that our mission is to support young scientists. More about why credit is mistakenly given to Einstein for relativity theory is discussed by Logunov Logunov (2004).

Einstein was not the only one who ignored the work of Bachelier; Paul Levy did so as well. Paul Levy was considered a pioneer and authority on stochastic processes during Bachelier's time, although Bruno de Finetti introduced a dual concept of infinite divisibility in 1929, before the works of Levy in the early 1930s on this topic. Levy never mentioned the work of the obscure and little known mathematician Bachelier. The first to give credit to Bachelier was Kolmogorov in his 1931 paper Kolmogoroff (1931) (Russian translation A. N. Kolmogorov (1938) and English translation Shirayev (1992)). Later L.J. Savage translated Bachelier's work to English and showed it to Paul Samuelson. Samuelson extended the work of Bachelier by considering the log-returns rather than absolute numbers, popularized the work of Bachelier among economists and the translation of Bachelier's thesis was finally published in English in 1964 Cootner (1967). Many economists who extended the work of Bachelier won Nobel prizes, including Eugene Fama known for work on the efficient markets hypothesis, Paul Samuelson, and Myron Scholes for the Black-Scholes model, as well as Robert Merton.

Originally developed to model financial markets by Louis Bachelier in 1900, Brownian Motion applies to biology (biomolecule movement), environmental science (diffusion of pollutants), and mathematics (stochastic calculus).

Example 7.1 (Yahoo Stock Price Simulation). Investing in volatile stocks can be very risky. The Internet stocks during the late 1990's were notorious for their volatility. For example, the leading Internet stock Yahoo! started 1999 at \$62, rose to \$122, then fell back to \$55 in August, only to end the year at \$216. Even more remarkable is the fact that by January 2000, Yahoo! has risen more than 100-fold from its offering price of \$1.32 on April 15, 1996. In comparison, the Nasdaq 100, a benchmark market index, was up about 5-fold during the same period.

Stock prices fluctuate somewhat randomly. Maurice Kendall, in his seminal 1953 paper on the random walk nature of stock and commodity prices, observed that “*The series looks like a wandering one, almost as if once a week the Demon of Chance drew a random number from a symmetrical population of fixed dispersion and added to it the current price to determine next week’s price (p. 87).*” While a pure random walk model for Yahoo!‘s stock price is in fact not reasonable since its price cannot fall below zero, an alternative model that appears to provide reasonable results assumes that the logarithms of price changes, or returns, follow a random walk. This alternative model is the basis for the results in this example.

To evaluate a stock investment, we take the initial price as X_0 and then we need to determine what the stock price might be in year T , namely X_T . Our approach draws from the Black-Scholes Model for valuing stock options. Technically, the Black-Scholes Model assumes that X_T is determined by the solution to a stochastic differential equation. This leads to the *Geometric Brownian Motion*

$$X_T = X_0 \exp((\mu - 1/2\sigma^2)T + \sigma B_T),$$

where B_T is a standard Brownian motion; that is, $B_0 = 0$, $B_t - B_s$ is independent of B_s , and its distribution depends only on $t - s$ with $B_t \sim N(0, t)$. Hence, $B_t = \sqrt{t}Z$, where $Z \sim N(0, 1)$.

Then, the expected value is

$$\begin{aligned} E(X_T) &= X_0 \exp((\mu - 1/2\sigma^2)T) E(\exp(\sigma B_T)) \\ &= X_0 \exp((\mu - 1/2\sigma^2)T) E(\exp(\sigma\sqrt{T}Z)) \\ &= X_0 \exp((\mu - 1/2\sigma^2)T) E(\exp(\sigma\sqrt{T}Z)) \\ &= X_0 \exp((\mu - 1/2\sigma^2)T) \exp\left(\frac{1}{2}\sigma^2 T\right) = X_0 \exp(\mu T). \end{aligned}$$

The $E(\exp(\sigma\sqrt{T}Z)) = \exp(1/2\sigma^2 T)$ is due to the moment property of the log-normal distribution. This follows from the property that the expectation of a log-normal variable e^Y where $Y \sim N(\mu, \sigma^2)$ is $e^{\mu+\sigma^2/2}$. We can interpret μ as the expected rate of return

$$\hat{\mu} = \frac{1}{T} \log\left(\frac{X_T}{X_0}\right).$$

This provides a way to estimate the expected rate of return from the expected value of the stock price at time T , by plugging in the observed values of X_0 and X_T .

The variance is

$$\begin{aligned}\text{Var}(X_T) &= X_0^2 \exp(2(\mu - 1/2\sigma^2)T) \text{Var}(\exp(\sigma B_T)) \\ &= X_0^2 \exp(2(\mu - 1/2\sigma^2)T) \text{Var}(\exp(\sigma\sqrt{T}Z)) \\ &= X_0^2 \exp(2(\mu - 1/2\sigma^2)T) \exp(\sigma^2 T) - X_0^2 \exp(2(\mu - 1/2\sigma^2)T) \\ &= X_0^2 \exp(2\mu T) (\exp(\sigma^2 T) - 1).\end{aligned}$$

The important consequence of the model for predicting future prices is that $\log(X_T/X_0)$ has a normal distribution with mean $(\mu - \frac{1}{2}\sigma^2)T$ and variance $\sigma^2 T$ which is equivalent to saying that the ratio X_T/X_0 has a log-normal distribution. It is interesting that although the Black-Scholes result is a standard tool for valuing options in finance the log-normal predictive distribution that follows from its assumptions is not commonly studied. In order to forecast X_T we need to estimate the unknowns μ and σ (recall X_0 is known). The unknown parameters μ and σ can be interpreted as the instantaneous expected rate of return and the volatility, respectively. The mean parameter μ is known as the expected rate of return because the expected value of X_T is $X_0 e^{\mu T}$. There are a number of ways of estimating the unknown parameters. One approach is to use an equilibrium model for returns, such as the Capital Asset Pricing Model or CAPM. We will discuss this model later. Another approach is to use historical data to estimate the parameters. For example, the expected rate of return can be estimated as the average historical return. The volatility can be estimated as the standard deviation of historical returns. The Black-Scholes model is a continuous time model, but in practice we use discrete time data. The Black-Scholes model can be adapted to discrete time by replacing the continuous time Brownian motion with a discrete time random walk.

7.1.1 Optimal Portfolio Allocation: The Merton Rule

Given Geometric Brownian Motion for asset prices, a natural question arises: what fraction of wealth should an investor allocate to the risky asset to maximize long-run growth? This is the continuous-time analog of the Kelly criterion introduced in Chapter 3 for discrete bets.

Consider an investor who can allocate fraction f to a risky asset following GBM with expected return μ and volatility σ , with the remainder in a risk-free asset. The investor's wealth W_t evolves as:

$$W_T = W_0 \exp((f\mu - \frac{1}{2}f^2\sigma^2)T + f\sigma B_T)$$

Maximizing the expected log-growth rate $E(\log(W_T/W_0))/T$ yields the **Merton rule**:

$$f^* = \frac{\mu}{\sigma^2} \tag{7.1}$$

where μ is the expected excess return (risk premium) over the risk-free rate and σ^2 is the variance. This is precisely the Kelly criterion in the log-normal limit.

The intuition is clear: allocate more to the risky asset when the expected return is high and less when volatility is high. The ratio μ/σ^2 balances reward against risk, measured by variance rather than standard deviation because variance compounds geometrically over time.

In practice, the full Merton allocation can be aggressive. The **fractional Merton** (or fractional Kelly) strategy scales down by a risk-aversion parameter:

$$f^* = \frac{1}{\gamma} \frac{\mu}{\sigma^2}$$

where γ is the coefficient of relative risk aversion from CRRA utility $U(W) = W^{1-\gamma}/(1-\gamma)$. Setting $\gamma = 1$ recovers log utility and full Merton; $\gamma = 2$ gives “Half-Kelly.”

Example 7.2 (Merton Rule for S&P 500). Historically, the S&P 500 has had an annual excess return of approximately $\mu \approx 0.06$ (6%) and volatility $\sigma \approx 0.16$ (16%). The Merton rule suggests:

$$f^* = \frac{0.06}{0.16^2} = \frac{0.06}{0.0256} \approx 2.3$$

This implies leveraging 230% into equities—borrowing 130% and investing 230%! Such aggressive positioning explains why practitioners often use fractional Kelly with $\gamma = 2$ or $\gamma = 3$, yielding allocations of 115% or 77% respectively.

7.2 Black-Scholes Model for Sports Betting

Sports betting involves wagering on the outcome of athletic events. Bettors’ assessments of these outcomes are aggregated in markets that provide key metrics like the point spread, which is the expected margin of victory, and moneyline odds, which imply the probability of a team winning. These market-based measures can be used to analyze the uncertainty, or volatility, inherent in a sports game.

To quantify the uncertainty in a game’s outcome, the score difference between two teams over time can be modeled as a stochastic process. Specifically, we use a Brownian motion model, first proposed by H. S. Stern (1994), to represent the evolution of a team’s lead. In this framework, the score difference at time

t , denoted as X_t , is assumed to follow a normal distribution with a mean (or “drift”) that grows over time and a variance that also increases with time.

This can be expressed mathematically as:

$$X_t = \mu t + \sigma B_t \sim N(\mu t, \sigma^2 t)$$

where μ is the drift parameter, representing the favored team’s point advantage over the whole game (derived from the point spread), σ is the volatility parameter, representing the standard deviation of the final outcome, and t is the time elapsed in the game, scaled from 0 to 1. We normalize the game duration to $T = 1$.

This model allows for the calculation of a team’s win probability at any point in the game and provides a formal way to measure the uncertainty of the final score.

The concept of deriving a game’s volatility from betting markets is directly analogous to the Black-Scholes model in finance. In finance, the Black-Scholes formula is used to price options. If the market price of an option is known, one can work backward to solve for the volatility of the underlying stock; this is called *implied volatility*. The model in sports betting does the same: it uses the market-set point spread (μ) and win probability (p) to solve for the game’s implied volatility (σ).

Both models use a Brownian motion framework to describe how a variable changes over time. However, there is a key difference. The sports model uses a standard Brownian motion, where the score changes additively. In contrast, the Black-Scholes model uses a geometric Brownian motion, which assumes that a stock price changes by a certain percentage, not by a fixed amount.

Essentially, this approach applies the financial concept of implied volatility to the sports world, creating a lens through which betting market data can be interpreted to measure the expected uncertainty of a game.

7.2.1 Implied Volatility for Sports Games

The concept of *implied volatility* is central to understanding how market prices reflect uncertainty. In the context of sports betting, implied volatility represents the market’s assessment of the uncertainty in a game’s final outcome, derived from observable betting market data.

Given the point spread μ (which represents the expected margin of victory) and the win probability p (derived from moneyline odds), we can solve for the implied volatility σ using the relationship:

$$p = \Phi\left(\frac{\mu}{\sigma}\right)$$

Rearranging this equation, the implied volatility is given by:

$$\sigma = \frac{\mu}{\Phi^{-1}(p)}$$

where Φ^{-1} is the inverse of the standard normal cumulative distribution function (the quantile function).

This approach mirrors the methodology used in financial markets, where option prices are used to infer the market's expectation of future stock price volatility. In sports betting, the "option price" is effectively the betting odds, and the "underlying asset" is the game outcome. Just as financial implied volatility reflects market sentiment about future price movements, sports implied volatility captures the market's view of how uncertain or "volatile" a particular game is likely to be.

For example, a game between two closely matched teams might have high implied volatility, reflecting greater uncertainty in the outcome, while a game featuring a heavily favored team against a significant underdog would typically exhibit lower implied volatility, as the outcome is more predictable.

Example 7.3 (Black-Scholes Model for Super Bowl). In order to define the implied volatility of a sports game we begin with a distributional model for the evolution of the outcome in a sports game which we develop from Stern (1994). The model specifies the distribution of the lead of team A over team B, X_t for any t as a Brownian motion process. If B_t denotes a standard Brownian motion with distributional property $B_t \sim N(0, t)$ and we incorporate drift, μ , and volatility, σ , terms, then the evolution of the outcome X_t that is given by:

$$X_t = \mu t + \sigma B_t \sim N(\mu t, \sigma^2 t).$$

This distribution of the game outcome is similar to the Black-Scholes model of the distribution of a stock price.

This specification results in several useful measures (or, this specification results in closed-form solutions for a number of measures of interest). The distribution of the final score follows a normal distribution, $X_1 \sim N(\mu, \sigma^2)$. We can calculate the probability of team A winning, denoted $p = \mathbb{P}(X_1 > 0)$, from the spread and probability distribution. Given the normality assumption, $X_1 \sim N(\mu, \sigma^2)$, we have

$$p = \mathbb{P}(X_1 > 0) = \Phi\left(\frac{\mu}{\sigma}\right)$$

where Φ is the standard normal cdf. Table 7.1 uses Φ to convert team A's advantage μ to a probability scale using the information ratio μ/σ .

Table 7.1: Probability of Winning p versus the Sharpe Ratio μ/σ (measure of risk-adjusted return)

μ/σ	0	0.25	0.5	0.75	1	1.25	1.5	2
$p = \Phi(\mu/\sigma)$	0.5	0.60	0.69	0.77	0.84	0.89	0.93	0.977

If teams are evenly matched and $\mu/\sigma = 0$ then $p = 0.5$. Table 7.1 provides a list of probabilities as a function of μ/σ . For example, if the point spread $\mu = -4$ and volatility is $\sigma = 10.6$, then the team has a $\mu/\sigma = -4/10.6 = -0.38$ volatility point disadvantage. The probability of winning is $\Phi(-0.38) = 0.353 < 0.5$. A common scenario is that team A has an edge equal to half a volatility, so that $\mu/\sigma = 0.5$ and then $p = 0.69$.

Of particular interest here are conditional probability assessments made as the game progresses. For example, suppose that the current lead at time t is l points and so $X_t = l$. The model can then be used to update your assessment of the distribution of the final score with the conditional distribution $(X_1|X_t = l)$. To see this, we can re-write the distribution of X_1 given X_t by noting that $X_1 = X_t + X_1 - X_t$. Using the formula above and substituting t for 1 where appropriate and noting that $X_t = l$ by assumption, this simplifies to

$$X_1 = l + \mu(1-t) + \sigma(B(1) - B_t).$$

Here $B(1) - B_t \stackrel{D}{=} B(1-t)$ which is independent of X_t with distribution $N(0, 1-t)$. The mean and variance of $X_1|X_t = l$ decay to zero as $t \rightarrow 1$ and the outcome becomes certain at the realised value of X_1 . We leave open the possibility of a tied game and overtime to determine the outcome.

To determine this conditional distribution, we note that there are $1-t$ time units left together with a drift μ and as shown above in this case the uncertainty can be modeled as $\sigma^2(1-t)$. Therefore, we can write the distribution of the final outcome after t periods with a current lead of l for team A as the conditional distribution:

$$(X_1|X_t = l) = (X_1 - X_t) + l \sim N(l + \mu(1-t), \sigma^2(1-t))$$

From the conditional distribution $(X_1|X_t = l) \sim N(l + \mu(1-t), \sigma^2(1-t))$, we can calculate the conditional probability of winning as the game evolves. The probability of team A winning at time t given a current lead of l point is:

$$p_t = P(X_1 > 0|X_t = l) = \Phi\left(\frac{l + \mu(1-t)}{\sigma\sqrt{1-t}}\right)$$

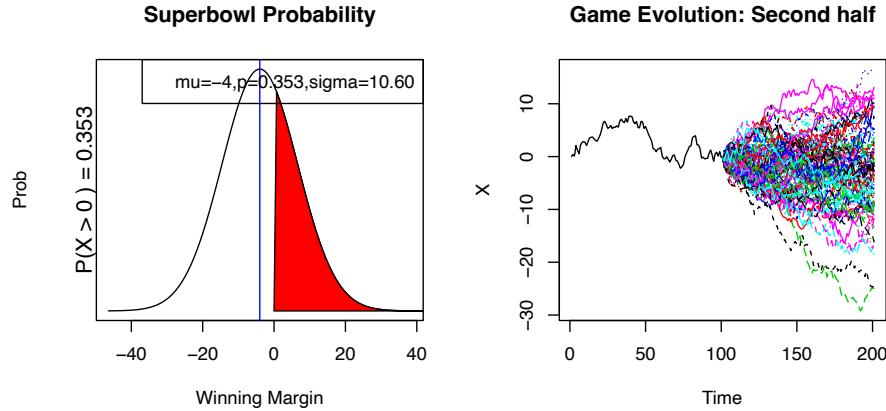


Figure 7.2: Score Evolution on a Discretized Grid

Figure 7.2 A and B illustrate our methodology with an example. Suppose we are analyzing data for a Superbowl game between teams A and B with team A favored. Figure A presents the information available at the beginning of the game from the perspective of the underdog team B. If the initial point spread—or the market's expectation of the expected outcome—is -4 and the volatility is 10.6 (assumed given for the moment; more on this below) then the probability that the underdog team wins is $p = \Phi(\mu/\sigma) = \Phi(-4/10.6) = 35.3\%$. This result relies on our assumption of a normal outcome distribution on the outcome as previously explained. Another way of saying this is $\mathbb{P}(X(1) > 0) = 0.353$ for an outcome distribution $X(1) \sim N(-4, 10.6^2)$. Figure A illustrates this with the shaded red area under the curve.

Figure 7.2 B illustrates the information and potential outcomes at half-time. Here we show the evolution of the actual score until half time as the solid black line. From half-time onwards we simulate a set of possible Monte Carlo paths to the end of the game.

Specifically, we discretise the model with time interval $\Delta = 1/200$ and simulate possible outcomes given the score at half time. The volatility plays a key role in turning the point spread into a probability of winning as the greater the volatility of the distribution of the outcome, X_1 , the greater the range of outcomes projected in the Monte Carlo simulation. Essentially the volatility provides a scale which calibrates the advantage implied by a given point spread.

We can use this relationship to determine how volatility decays over the course of the game. The conditional distribution of the outcome given the score at time t , is $(X_1 | X_t = l)$ with a variance of $\sigma^2(1-t)$ and volatility of $\sigma\sqrt{1-t}$. The volatility is a decreasing function of t , illustrating that the volatility dissipates

over the course of a game. For example, if there is an initial volatility of $\sigma = 10.6$, then at half-time when $t = \frac{1}{2}$, the volatility is $10.6/\sqrt{2} = 7.5$ volatility points left. Table 7.2, below, illustrates this relationship for additional points over the game.

Table 7.2: Volatility Decay over Time

t	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1
$\sigma\sqrt{1-t}$	10.6	9.18	7.50	5.3	0

To provide insight into the final outcome given the current score, Table 7.1 and Table 7.2 can be combined to measure the current outcome, l , in terms of standard deviations of the outcome.

For example, suppose that you have Team B, an underdog, so from their perspective $\mu = -4$ and at half-time team B has a lead of 15, $l = 15$. Team B's expected outcome as presented earlier is $l + \mu(1-t)$ or $15 - 4 \times \frac{1}{2} = 13$. If initial volatility is $\sigma = 10.60$ then the remaining volatility at half-time is $10.6/\sqrt{2} = 7.50$ and team B's expected outcome of 13 in terms of standard deviations is $13/7.5 = 1.73$. Thus team B's expected outcome is at the 99th percentile of the distribution, $\Phi(1.73) = 0.96$, implying a 96% chance of winning.

Implied Volatility

The previous discussion assumed that the variance (or volatility) parameter σ was a known constant. We return to this important quantity now. We are now in a position to define the *implied volatility* implicit in the two betting lines that are available. Given our model, we will use the *money-line* odds to provide a market assessment of the probability of winning, p , and the *point spread* to assess the expected margin of victory, μ . The money line odds are shown for each team A and B and provide information on the payoff from a bet on the team winning. This calculation will also typically require an adjustment for the bookmaker's spread. With these we can infer the *implied volatility*, σ_{IV} , by solving

$$\sigma_{IV} : \quad p = \Phi\left(\frac{\mu}{\sigma_{IV}}\right) \quad \text{which gives } \sigma_{IV} = \frac{\mu}{\Phi^{-1}(p)}.$$

Here $\Phi^{-1}(p)$ denotes the standard normal quantile function such that the area under the standard normal curve to the left of $\Phi^{-1}(p)$ is equal to p . In our example we calculate this using the `qnorm` in R. Note that when $\mu = 0$ and $p = \frac{1}{2}$ there's no market information about the volatility as $\mu/\Phi^{-1}(p)$ is undefined. This is the special case where the teams are seen as evenly matched—the expected outcome has a zero point spread and there is an equal probability that either team wins.

Time Varying Implied Volatility

Up to this point the volatility rate has been assumed constant through the course of the game, i.e., that the same value of σ is relevant. The amount of volatility remaining in the game is not constant but the basic underlying parameters has been assumed constant. This need not be true and more importantly the betting markets may provide some information about the best estimate of the volatility parameter at a given point of time. This is important because time-varying volatility provides an interpretable quantity that can allow one to assess the value of a betting opportunity.

With the advent of online betting there is a virtually continuous traded contract available to assess implied expectations of the probability of team A winning at any time t . The additional information available from the continuous contract allows for further update of the implied conditional volatility. We assume that the online betting market gives us a current assessment of p_t , that is the current probability that team A will win. We will then solve for σ^2 and in turn define resulting time-varying volatility, as $\sigma_{IV,t}$, using the resulting equation to solve for $\sigma_{IV,t}$ with

$$p_t = \Phi \left(\frac{l + \mu(1-t)}{\sigma_{IV,t}\sqrt{1-t}} \right) \quad \text{which gives } \sigma_{IV,t} = \frac{l + \mu(1-t)}{\Phi^{-1}(p_t)\sqrt{1-t}}$$

We will use our methodology to find evidence of time-varying volatility in the SuperBowl XLVII probabilities.

Super Bowl XLVII: Ravens vs San Francisco 49ers

Super Bowl XLVII was held at the Superdome in New Orleans on February 3, 2013 and featured the San Francisco 49ers against the Baltimore Ravens. Going into Super Bowl XLVII the San Francisco 49ers were favorites to win which was not surprising following their impressive season. It was a fairly bizarre Super Bowl with a 34 minute power outage affecting the game by ultimately an exciting finish with the Ravens causing an upset victory 34–31. We will build our model from the viewpoint of the Ravens. Hence X_t will correspond to the Raven's score minus the San Francisco 49ers. Table 7.3 provides the score at the end of each quarter.

Table 7.3: SuperBowl XLVII by Quarter

t	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1
Ravens	0	7	21	28	34
49ers	0	3	6	23	31
X_t	0	4	15	5	3

To determine the parameters of our model we first use the *point spread* which was set at the Ravens being a four point underdog, i.e. $\mu = -4$. This sets the

mean of our outcome, X_1 , as

$$\mu = \mathbb{E}(X_1) = -4.$$

In reality, it was an exciting game with the Ravens upsetting the 49ers by $34 - 31$. Hence, the realised outcome is $X_1 = 34 - 31 = 3$ with the point spread being beaten by 7 points or the equivalent of a touchdown.

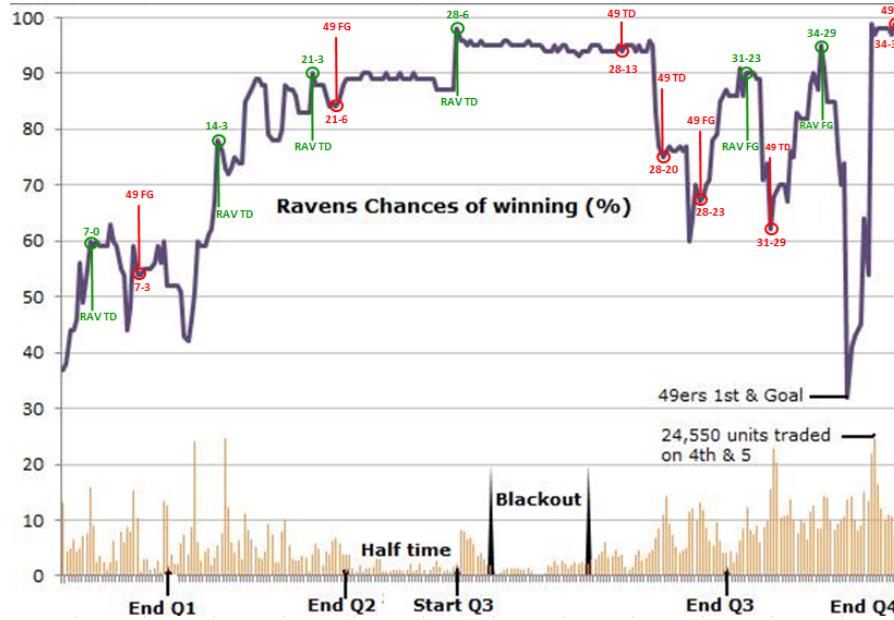


Figure 7.3: Superbowl XLVII: Ravens vs 49ers: TradeSports contracts traded and dynamic probability of the Ravens winning

To determine the markets' assessment of the probability that the Ravens would win at the beginning of the game we use the *money-line* odds. These odds were quoted as San Francisco -175 and Baltimore Ravens $+155$. This implies that a bettor would have to place \$175 to win \$100 on the 49ers and a bet of \$100 on the Ravens would lead to a win of \$155. We can convert both of these money-lines to *implied probabilities* of the each team winning, by the equations

$$p_{SF} = \frac{175}{100 + 175} = 0.686 \quad \text{and} \quad p_{Ravens} = \frac{100}{100 + 155} = 0.392$$

The probability sum to one plus the market overround:

$$p_{SF} + p_{Ravens} = 0.686 + 0.392 = 1.078$$

namely a 7.8% edge for the bookmakers. Put differently, if bettors place money proportionally across both teams then the bookies *vig* will be

$$\text{Vig} = \frac{0.078}{0.078 + 1} = 0.072$$

This means that the bookmaker is expected to make a profit of 7.2% of the total amount staked, no matter what happens to the outcome of the game.

To account for this edge in our model, we use the mid-point of the spread to determine p implying that

$$p = \frac{1}{2}p_{Ravens} + \frac{1}{2}(1 - p_{SF}) = 0.353$$

From the Ravens perspective we have $p = \mathbb{P}(X_1 > 0) = 0.353$.

Figure 7.3 shows the evolution of the markets conditional probability of winning p_t for the Ravens. The data are from the online betting website [TradeSports.com](#). Starting at $p = 0.353$ we see how dramatically the markets assessment of the Ravens winning can fluctuate. Given their commanding lead at half time, the probability has as high as 0.90. At the end of the four quarter when the 49ers nearly went into the lead with a touchdown, at one point the probability had dropped to 30%.

Our main question of interest is then: *What implied volatility is consistent with market expectations?*

To calculate the implied volatility of the Superbowl we substitute the pair (μ, p) into our definition and solve for σ_{IV} . We obtain

$$\sigma_{IV} = \frac{\mu}{\Phi^{-1}(p)} = \frac{-4}{-0.377} = 10.60$$

where we have used $\Phi^{-1}(p) = qnorm(0.353) = -0.377$. So on a volatility scale the 4 point advantage assessed for the 49ers is under a $\frac{1}{2}\sigma$ favorite. From Table 2, this is consistent with a win probability of $p = \Phi(\frac{1}{2}) = 0.69$. Another feature is that a $\sigma = 10.6$ is historically low, as a typical volatility of an NFL game is 14 (see Stern, 1991). This implies a standard deviation of the final score differential of approximately 10.6 points. However, the more competitive the game one might expect a lower volatility. In reality, the outcome $X_1 = 3$ was within one standard deviation of the model, which had an expectation of $\mu = -4$ and volatility $\sigma = 10.6$. Another question of interest is

What's the probability of the Ravens winning given their lead at half time?

At half-time the Ravens were leading 21 to 6. This gives us $X(\frac{1}{2}) = 21 - 6 = 15$. From the online betting market we also have traded contracts on [TradeSports.com](#) that yield a current probability of $p_{\frac{1}{2}} = 0.90$.

An alternative view is to assume that the market assesses time varying volatility and the prices fully reflect the underlying probability. Here we ask the question

What's the implied volatility for the second half of the game?

We now have an implied volatility

$$\sigma_{IV,t=\frac{1}{2}} = \frac{l + \mu(1-t)}{\Phi^{-1}(p_t)\sqrt{1-t}} = \frac{15 - 2}{\Phi^{-1}(0.9)/\sqrt{2}} = 14$$

where $\text{qnorm}(0.9)=1.28$. Notice that $14 > 10.6$, our assessment of the implied volatility at the beginning of the game.

What's a valid betting strategy?

An alternative approach is to assume that the initial moneyline and point spread set the volatility and this stays constant throughout the game. This market is much larger than the online market and this is a reasonable assumption unless there has been material information as the game progresses such as a key injury.

Hence the market was expected a more typical volatility in the second half. If a bettor believed that the volatility was closer to the historical average of $\sigma = 14$ (Stern, 1991), rather than the implied 10.6, then their assessment of the Ravens win probability would be lower. The remaining volatility would be $14/\sqrt{2} \approx 9.9$, yielding a Z-score of $13/9.9 \approx 1.31$. This corresponds to a win probability of $\Phi(1.31) \approx 0.905$. Compared to the market probability of 0.90, this represents a small edge.

The Kelly criterion (Kelly, 1956) yields the optimal betting fraction f^* :

$$f^* = p - \frac{q}{b} = 0.905 - \frac{0.095}{1/9} \approx 0.05$$

where $q = 1 - p$ is the probability of losing, and b is the odds received (net fractional odds). That is, 5% of capital. A more realistic strategy is to use the fractional Kelly criterion, which scales the bet by a risk-aversion parameter γ . For example, in this case if $\gamma = 3$, we would bet $0.05/3 \approx 0.017$, or 1.7% of our capital on this betting opportunity.

Finally, odds changes can be dramatic at the end of the fourth quarter, and this Super Bowl was no exception. With the score at 34–29 and only a few minutes remaining, the 49ers were at first-and-goal. A few minutes after this, the probability of the Ravens winning had dropped precipitously from over 90% to 30%, see Figure 7.3. On San Francisco's final offensive play of the game, Kaepernick threw a pass on fourth down to Michael Crabtree, but Ravens cornerback Jimmy Smith appeared to hold the wide receiver during the incompletion, No call was given and the final result was a Ravens win.

7.3 Poisson Process

A Poisson process models random events occurring independently at a constant average rate—customer arrivals, call center traffic, or goals in a soccer match. A counting process $\{N_t, t \geq 0\}$ is a Poisson process with rate $\lambda > 0$ if:

1. $N(0) = 0$ (the process starts at zero)
2. The process has independent increments: for any $0 \leq t_1 < t_2 < \dots < t_n$, the random variables $N(t_2) - N(t_1), N(t_3) - N(t_2), \dots, N(t_n) - N(t_{n-1})$ are independent
3. The process has stationary increments: for any $s < t$, the distribution of $N_t - N(s)$ depends only on the length of the interval $t - s$
4. For any interval of length t , the number of events follows a Poisson distribution:

$$P(N_t = k) = \frac{e^{-\lambda t} (\lambda t)^k}{k!}, \quad k = 0, 1, 2, \dots$$

The parameter λ represents the rate at which events occur per unit time. The expected number of events in an interval of length t is $E(N_t) = \lambda t$, and the variance is $\text{Var}(N_t) = \lambda t$.

Figure 7.4 below shows three sample paths of a Poisson process with rate $\lambda = 5$ events per unit time.

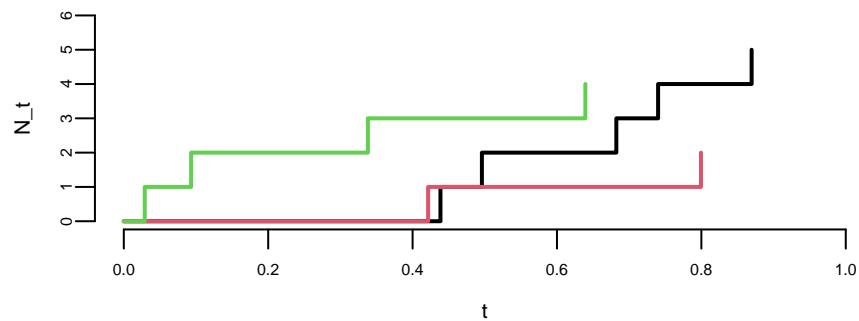


Figure 7.4: Poisson Process Trajectories

An equivalent characterization of the Poisson process is through the inter-arrival times between consecutive events. If T_1, T_2, \dots denote the times between successive events, then these are independent and identically distributed exponential random variables with mean $1/\lambda$. This connection between the Poisson and exponential distributions is fundamental: the Poisson process counts

events while the exponential distribution models the waiting time between events.

The Poisson process can be viewed from two complementary perspectives. From a *continuous-time* viewpoint, we track the evolution of the counting process N_t as time progresses, asking questions about the probability of observing a certain number of events by time t or the distribution of event times. From a *discrete count data* perspective, we observe the number of events that occurred during a fixed time interval and use this to make inferences about the underlying rate parameter λ .

Chapter 3 introduced Poisson models in the context of count data and Bayesian inference. The Poisson distribution (discussed in the section on Poisson Model for Count Data) emerges naturally when we observe a Poisson process over a fixed time interval. For instance, when modeling the number of goals scored by a soccer team in a match, we implicitly assume that goals occur according to a Poisson process with some rate λ , and we observe the total count at the end of the match.

The Bayesian approach to learning about the rate parameter λ (covered in the section on Poisson-Gamma: Learning about a Poisson Intensity in Chapter 3) becomes particularly powerful in the continuous-time setting. When we observe a Poisson process over time, we can update our beliefs about λ as new events occur. The Gamma distribution serves as a conjugate prior for λ , meaning that if we start with a Gamma prior and observe events from a Poisson process, the posterior distribution remains in the Gamma family with updated parameters. This elegant updating mechanism allows us to refine our estimates of the event rate as we gather more data, balancing prior beliefs with observed evidence.

The connection between these perspectives is crucial for applications. In many real-world scenarios, we observe event counts over fixed intervals (discrete perspective) but need to make predictions about future events or the timing of the next event (continuous perspective). The Poisson process framework unifies these views, allowing us to seamlessly move between counting events and modeling their temporal dynamics.

Example 7.4 (EPL Betting). Feng, Polson, and Xu (2016) employ a Skellam process (a difference of Poisson random variables) to model real-time betting odds for English Premier League (EPL) soccer games. Given a matrix of market odds on all possible score outcomes, we estimate the expected scoring rates for each team. The expected scoring rates then define the implied volatility of an EPL game. As events in the game evolve, they re-estimate the expected scoring rates and our implied volatility measure to provide a dynamic representation of the market's expectation of the game outcome. They use real-time market odds data for a game between Everton and West Ham in the 2015-2016 season. We show how the implied volatility for the outcome evolves as goals, red cards, and corner kicks occur.

Gambling on soccer is a global industry with revenues of over \$1 trillion a year (see “Football Betting - the Global Gambling Industry worth Billions,” BBC Sport). Betting on the result of a soccer match is a rapidly growing market, and online real-time odds exist (Betfair, Bet365, Ladbrokes). Market odds for all possible score outcomes ($0 - 0, 1 - 0, 0 - 1, 2 - 0, \dots$) as well as outright win, lose, and draw are available in real time. In this paper, we employ a two-parameter probability model based on a Skellam process and a non-linear objective function to extract the expected scoring rates for each team from the odds matrix. The expected scoring rates then define the implied volatility of the game.

Skellam Process

To model the outcome of a soccer game between team A and team B, we let the difference in scores, $N_t = N_{A,t} - N_{B,t}$, where $N_{A,t}$ and $N_{B,t}$ are the team scores at time point t . Negative values of N_t indicate that team A is behind. We begin at $N(0) = 0$ and end at time one with N_1 representing the final score difference. The probability $\mathbb{P}(N_1 > 0)$ represents the ex-ante odds of team A winning. Half-time score betting, which is common in Europe, is available for the distribution of $N_{1/2}$.

Then we find a probabilistic model for the distribution of N_1 given $N_t = \ell$, where ℓ is the current lead. This model, together with the current market odds, can be used to infer the expected scoring rates of the two teams and then to define the implied volatility of the outcome of the match. We let λ^A and λ^B denote the expected scoring rates for the whole game. We allow for the possibility that the scoring abilities (and their market expectations) are time-varying, in which case we denote the expected scoring rates after time t by λ_t^A and λ_t^B , respectively, instead of $\lambda^A(1-t)$ and $\lambda^B(1-t)$.

The Skellam distribution is defined as the difference between two independent Poisson variables given by:

$$\begin{aligned} N_{A,t} &= W_{A,t} + W_t \\ N_{B,t} &= W_{B,t} + W_t \end{aligned}$$

where $W_{A,t}$, $W_{B,t}$, and W_t are independent processes with:

$$W_{A,t} \sim \text{Poisson}(\lambda^A t), \quad W_{B,t} \sim \text{Poisson}(\lambda^B t).$$

Here W_t is a Poisson process used to induce a correlation between the numbers of goals scored.

$$N_t = N_{A,t} - N_{B,t} \sim \text{Skellam}(\lambda^A t, \lambda^B t). \quad (7.2)$$

At time t , we have the conditional distributions:

$$\begin{aligned} W_A(1) - W_{A,t} &\sim \text{Poisson}(\lambda^A(1-t)) \\ W_B(1) - W_{B,t} &\sim \text{Poisson}(\lambda^B(1-t)). \end{aligned}$$

Now letting $N^*(1-t)$, the score difference of the sub-game which starts at time t and ends at time 1 and the duration is $(1-t)$. By construction, $N_1 = N_t + N^*(1-t)$. Since $N^*(1-t)$ and N_t are differences of two Poisson process on two disjoint time periods, by the property of Poisson process, $N^*(1-t)$ and N_t are independent. Hence, we can re-express equation (Equation 7.2) in terms of $N^*(1-t)$, and deduce

$$N^*(1-t) = W_A^*(1-t) - W_B^*(1-t) \sim \text{Skellam}(\lambda_t^A, \lambda_t^B)$$

where $W_A^*(1-t) = W_A(1) - W_{A,t}$, $\lambda^A = \lambda_0^A$ and $\lambda_t^A = \lambda^A(1-t)$. A natural interpretation of the expected scoring rates, λ_t^A and λ_t^B , is that they reflect the “net” scoring ability of each team from time t to the end of the game. The term W_t models a common strength due to external factors, such as weather. The “net” scoring abilities of the two teams are assumed to be independent of each other as well as the common strength factor. We can calculate the probability of any particular score difference, given by $\mathbb{P}(N_1 = x | \lambda^A, \lambda^B)$, at the end of the game where the λ ’s are estimated from the matrix of market odds. Team strength and “net” scoring ability can be influenced by various underlying factors, such as the offensive and defensive abilities of the two teams. The goal of our analysis is to only represent these parameters at every instant as a function of the market odds matrix for all scores.

Another quantity of interest is the conditional probability of winning as the game progresses. If the current lead at time t is ℓ , and $N_t = \ell = N_{A,t} - N_{B,t}$, the Poisson property implied that the final score difference ($N_1 | N_t = \ell$) can be calculated by using the fact that $N_1 = N_t + N^*(1-t)$ and N_t and $N^*(1-t)$ are independent. Specifically, conditioning on $N_t = \ell$, we have the identity

$$N_1 = N_t + N^*(1-t) = \ell + \text{Skellam}(\lambda_t^A, \lambda_t^B).$$

We are now in a position to find the conditional distribution ($N_1 = x | N_t = \ell$) for every time point t of the game given the current score. Simply put, we have the time homogeneous condition

$$\begin{aligned} \mathbb{P}(N_1 = x | \lambda_t^A, \lambda_t^B, N_t = \ell) &= \mathbb{P}(N_1 - N_t = x - \ell | \lambda_t^A, \lambda_t^B, N_t = \ell) \\ &= \mathbb{P}(N^*(1-t) = x - \ell | \lambda_t^A, \lambda_t^B) \end{aligned}$$

where $\lambda_t^A, \lambda_t^B, \ell$ are given by market expectations at time t . See Feng et al. for details.

Market Calibration

Our information set at time t includes the current lead $N_t = \ell$ and the market odds for $\{Win, Lose, Draw, Score\}_t$, where $Score_t = \{(i - j) : i, j = 0, 1, 2, \dots\}$. These market odds can be used to calibrate a Skellam distribution which has only two parameters λ_t^A and λ_t^B . The best fitting Skellam model with parameters $\{\hat{\lambda}_t^A, \hat{\lambda}_t^B\}$ will then provide a better estimate of the market's information concerning the outcome of the game than any individual market (such as win odds) as they are subject to a "vig" and liquidity.

Suppose that the fractional odds for all possible final score outcomes are given by a bookmaker. Fractional odds, commonly used in the UK, express the ratio of profit to stake. For example, odds of $3 : 1$ (read as "three-to-one") mean that for every \$1 wagered, the bettor receives \$3 in profit if the bet wins, plus the original \$1 stake returned, for a total payout of \$4. In this case, if the bookmaker offers $3 : 1$ odds on a 2-1 final score, the bookmaker pays out three times the amount staked by the bettor if the outcome is indeed 2-1. This contrasts with American money-line odds, where positive numbers indicate the profit on a \$100 stake (e.g., +300 means \$300 profit on \$100 wagered), and negative numbers indicate the stake needed to win \$100.

The market implied probability makes the expected winning amount of a bet equal to 0. For fractional odds of $3 : 1$, the implied probability is calculated as $p = \frac{1}{1+3} = \frac{1}{4} = 0.25$ or 25%. We can verify this creates a fair bet: the expected winning amount is $\mu = -1 \times (1 - 1/4) + 3 \times (1/4) = -0.75 + 0.75 = 0$. We denote these odds as $odds(2, 1) = 3$. To convert all the available odds to implied probabilities, we use the identity

$$\mathbb{P}(N_A(1) = i, N_B(1) = j) = \frac{1}{1 + odds(i, j)}.$$

The market odds matrix, O , with elements $o_{ij} = odds(i-1, j-1)$, $i, j = 1, 2, 3, \dots$ provides all possible combinations of final scores. Odds on extreme outcomes are not offered by the bookmakers. Since the probabilities are tiny, we set them equal to 0. The sum of the possible probabilities is still larger than 1 (see M. J. Dixon and Coles (1997) and M. J. Dixon and Coles (1997)). This "excess" probability corresponds to a quantity known as the "market vig." For example, if the sum of all the implied probabilities is 1.1, then the expected profit of the bookmaker is 10%. To account for this phenomenon, we scale the probabilities to sum to 1 before estimation.

To estimate the expected scoring rates, λ_t^A and λ_t^B , for the sub-game $N^*(1-t)$, the odds from a bookmaker should be adjusted by $N_{A,t}$ and $N_{B,t}$. For example, if $N_A(0.5) = 1$, $N_B(0.5) = 0$ and $odds(2, 1) = 3$ at half time, these observations actually says that the odds for the second half score being 1-1 is 3 (the outcomes for the whole game and the first half are 2-1 and 1-0 respectively, thus the outcome for the second half is 1-1). The adjusted $odds^*$

for $N^*(1-t)$ is calculated using the original odds as well as the current scores and given by

$$odds^*(x, y) = odds(x + N_{A,t}, y + N_{B,t}).$$

At time t ($0 \leq t \leq 1$), we calculate the implied conditional probabilities of score differences using odds information

$$\mathbb{P}(N_1 = k | N_t = \ell) = \mathbb{P}(N^*(1-t) = k - \ell) = \frac{1}{c} \sum_{i-j=k-\ell} \frac{1}{1 + odds^*(i, j)}$$

where $c = \sum_{i,j} \frac{1}{1+odds^*(i,j)}$ is a scale factor, $\ell = N_{A,t} - N_{B,t}$, $i, j \geq 0$ and $k = 0, \pm 1, \pm 2, \dots$

Example: Everton vs West Ham (3/5/2016)

Table below shows the implied Skellam probabilities.

Table 7.4: Table: Original odds data from Ladbrokes before the game started.

Everton	West Ham	0	1	2	3	4	5
0		11/1	12/1	28/1	66/1	200/1	450/1
1		13/2	6/1	14/1	40/1	100/1	350/1
2		7/1	7/1	14/1	40/1	125/1	225/1
3		11/1	11/1	20/1	50/1	125/1	275/1
4		22/1	22/1	40/1	100/1	250/1	500/1
5		50/1	50/1	90/1	150/1	400/1	
6		100/1	100/1	200/1	250/1		
7		250/1	275/1	375/1			
8		325/1	475/1				

Table 7.4 shows the raw data of odds right the game. We need to transform odds data into probabilities. For example, for the outcome 0-0, 11/1 is equivalent to a probability of 1/12. Then we can calculate the marginal probability of every score difference from -4 to 5. We neglect those extreme scores with small probabilities and rescale the sum of event probabilities to one.

Table 7.5: Market implied probabilities for the score differences versus Skellam implied probabilities at different time points. The estimated parameters $\hat{\lambda}^A = 2.33$, $\hat{\lambda}^B = 1.44$.

Score difference	-4	-3	-2	-1	0	1	2	3	4	5
Market Prob. (%)	1.70	2.03	4.88	12.33	21.93	22.06	16.58	9.82	4.72	2.23
Skellam Prob. (%)	0.78	2.50	6.47	13.02	19.50	21.08	16.96	10.61	5.37	2.27

Table 7.5 shows the model implied probability for the outcome of score differences before the game, compared with the market implied probability. As we see, the Skellam model appears to have longer tails. Different from independent Poisson modeling in M. J. Dixon and Coles (1997), our model is more flexible with the correlation between two teams. However, the trade-off of flexibility is that we only know the probability of score difference instead of the exact scores.

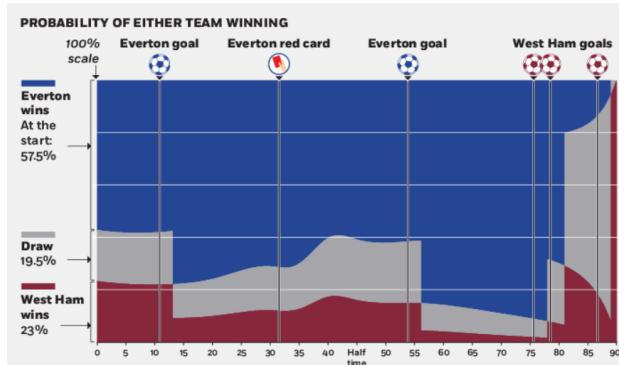


Figure 7.5: The betting market data for Everton and West Ham is from ladbrokes.com. Market implied probabilities (expressed as percentages) for three different results (Everton wins, West Ham wins and draw) are marked by three distinct colors, which vary dynamically as the game proceeds. The solid black line shows the evolution of the implied volatility. The dashed line shows significant events in the game, such as goals and red cards. Five goals in this game are 13' Everton, 56' Everton, 78' West Ham, 81' West Ham and 90' West Ham.

Figure 7.5 examines the behavior of the two teams and represent the market predictions on the final result. Notably, we see the probability change of win/draw/loss for important events during the game: goals scoring and a red

card penalty. In such a dramatic game, the winning probability of Everton gets raised to 90% before the first goal of West Ham in 78th minutes. The first two goals scored by West Ham in the space of 3 minutes completely reverses the probability of winning. The probability of draw gets raised to 90% until we see the last-gasp goal of West Ham that decides the game.

Figure 7.5 plots the path of implied volatility throughout the course of the game. Instead of a downward sloping line, we see changes in the implied volatility as critical moments occur in the game. The implied volatility path provides a visualization of the conditional variation of the market prediction for the score difference. For example, when Everton lost a player by a red card penalty at 34th minute, our estimates $\hat{\lambda}_t^A$ and $\hat{\lambda}_t^B$ change accordingly. There is a jump in implied volatility and our model captures the market expectation adjustment about the game prediction. The change in $\hat{\lambda}_A$ and $\hat{\lambda}_B$ are consistent with the findings of Vecer, Kopriva, and Ichiba (2009) where the scoring intensity of the penalized team drops while the scoring intensity of the opposing team increases. When a goal is scored in the 13th minute, we see the increase of $\hat{\lambda}_t^B$ and the market expects that the underdog team is pressing to come back into the game, an effect that has been well-documented in the literature. Another important effect that we observe at the end of the game is that as goals are scored (in the 78th and 81st minutes), the markets expectation is that the implied volatility increases again as one might expect.

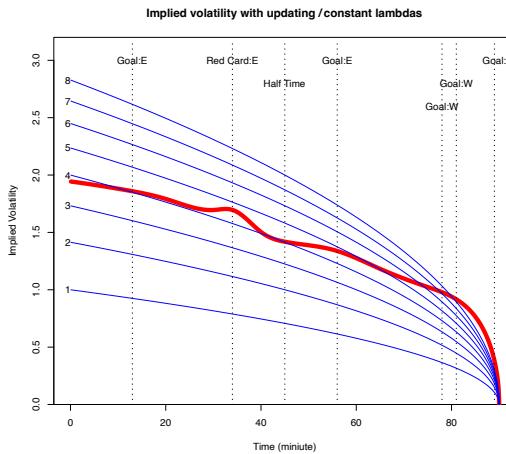


Figure 7.6: Red line: the path of implied volatility throughout the game, i.e., $\sigma_t^{red} = \sqrt{\hat{\lambda}_t^A + \hat{\lambda}_t^B}$. Blue lines: the path of implied volatility with constant $\lambda^A + \lambda^B$, i.e., $\sigma_t^{blue} = \sqrt{(\lambda^A + \lambda^B) * (1 - t)}$. Here $(\lambda^A + \lambda^B) = 1, 2, \dots, 8$.

Table 7.6: The calibrated $\{\hat{\lambda}_t^A, \hat{\lambda}_t^B\}$ divided by $(1-t)$ and the implied volatility during the game. $\{\lambda_t^A, \lambda_t^B\}$ are expected goals scored for rest of the game. The less the remaining time, the less likely to score goals. Thus $\{\hat{\lambda}_t^A, \hat{\lambda}_t^B\}$ decrease as t increases to 1. Dividing them by $(1-t)$ produces an updated version of $\hat{\lambda}_0$'s for the whole game, which are in general time-varying (but not decreasing necessarily).

t	0	0.11	0.22	0.33	0.44	0.50	0.61	0.72	0.83	0.94	1
$\hat{\lambda}_t^A/(1-t)$	2.33	2.51	2.53	2.46	1.89	1.85	2.12	2.12	2.61	4.61	0
$\hat{\lambda}_t^B/(1-t)$	1.44	1.47	1.59	1.85	2.17	2.17	2.56	2.90	3.67	5.92	0
$(\hat{\lambda}_t^A + \hat{\lambda}_t^B)/(1-t)$	3.78	3.98	4.12	4.31	4.06	4.02	4.68	5.03	6.28	10.52	0
$\sigma_{IV,t}$	1.94	1.88	1.79	1.70	1.50	1.42	1.35	1.18	1.02	0.76	0

Figure 7.6 compares the updating implied volatility of the game with implied volatilities of fixed $(\lambda^A + \lambda^B)$. At the beginning of the game, the red line (updating implied volatility) is under the “ $(\lambda^A + \lambda^B = 4)$ ”-blue line; while at the end of the game, it’s above the “ $(\lambda^A + \lambda^B = 8)$ ”-blue line. As we expect, the value of $(\hat{\lambda}_t^A + \hat{\lambda}_t^B)/(1-t)$ in Table 7.6 increases throughout the game, implying that the game became more and more intense and the market continuously updates its belief in the odds.

7.4 The Lévy-Itô Decomposition in Finance

One of the most profound results in the theory of stochastic processes is the Lévy-Itô decomposition theorem, which provides a universal framework for understanding how randomness evolves over time. The theorem states that any Lévy process (a stochastic process with stationary and independent increments) can be uniquely decomposed into three fundamental components:

1. A deterministic drift term (linear trend)
2. A continuous Gaussian component (Brownian motion)
3. A pure jump component (compound Poisson process)

Mathematically, any Lévy process X_t can be written as:

$$X_t = \mu t + \sigma B_t + J_t$$

where μ is the drift coefficient, B_t is standard Brownian motion with volatility

σ , and J_t represents the jump component that can be expressed as:

$$J_t = \sum_{i=1}^{N_t} Z_i$$

where N_t is a Poisson process counting the number of jumps up to time t , and Z_i are the jump sizes.

This decomposition is remarkable because it tells us that no matter how complex a stochastic process might appear, if it has independent and stationary increments, it can always be broken down into these three intuitive building blocks: a predictable trend, continuous random fluctuations, and discrete jumps.

The Lévy-Itô decomposition provides a natural motivation for studying Brownian motion and Poisson processes as fundamental objects. Brownian motion captures the continuous, infinitesimal random perturbations that accumulate over time, while the Poisson process models rare, discrete events that cause sudden changes in the system state. Together with a deterministic drift, these components form a complete toolkit for modeling virtually any phenomenon with independent increments.

The practical importance of this decomposition cannot be overstated. In finance, asset returns exhibit both continuous price movements (modeled by Brownian motion) and sudden jumps due to news announcements or market shocks (modeled by Poisson processes). In telecommunications, network traffic consists of a steady baseline load (drift) plus continuous fluctuations (Brownian component) and sudden spikes from large file transfers (jumps). In insurance, claim amounts follow a baseline trend with continuous variation and occasional catastrophic events.

Example 7.5 (Financial Asset Prices). Consider modeling the logarithm of a stock price. The Lévy-Itô decomposition suggests we should account for:

- *Drift*: The expected return on the asset, reflecting long-term growth trends in the economy
- *Brownian component*: Day-to-day price fluctuations driven by the continuous arrival of market information and trading activity
- *Jump component*: Sudden price movements triggered by earnings announcements, regulatory changes, or macroeconomic shocks

For instance, during the 2008 financial crisis, stock prices exhibited massive downward jumps that could not be explained by a pure Brownian motion model. The Lehman Brothers bankruptcy on September 15, 2008 caused the S&P 500 to drop by 4.7% in a single day—an event that would have probability essentially zero under a Gaussian model but is naturally accommodated by the jump component in the Lévy-Itô framework.

The practical application of this decomposition led to the development of jump-diffusion models in quantitative finance, where option prices are calculated by accounting for both continuous price movements and discrete jumps. This approach provides more realistic pricing and risk assessment compared to the classical Black-Scholes model, which assumes only continuous price movements.

The Lévy-Itô decomposition thus provides both theoretical insight and practical tools. It explains why Brownian motion and Poisson processes are the fundamental building blocks for continuous-time stochastic modeling, and it gives practitioners a principled framework for decomposing complex random phenomena into interpretable components that can be estimated, simulated, and managed separately.

7.5 Newton and the South Sea Bubble

The South Sea Bubble of 1720 stands as one of history's most spectacular financial disasters, demonstrating that even the greatest scientific minds can fall victim to speculative mania. The South Sea Company, established in 1711 ostensibly to trade with South America, proposed an audacious scheme: it would assume England's national debt in exchange for company shares and exclusive trading privileges. By early 1720, the company's directors launched an unprecedented campaign of stock manipulation, spreading rumors of fabulous wealth, bribing politicians and royalty, and offering generous credit terms that allowed investors to purchase shares with only a small down payment. The stock price soared from £128 at the start of 1720 to over £1,000 by June—an eightfold increase in six months (Figure 7.7). Sir Isaac Newton, then Master of the Royal Mint, initially profited by selling his holdings in April 1720 for £7,000, but he could not resist re-entering the market after watching shares continue to climb. When the bubble burst in September, Newton lost £20,000—several years of his salary—leading him to famously remark, “I can calculate the movement of stars, but not the madness of men.” His experience reveals how market psychology and the fear of missing out can overwhelm even the most disciplined rational minds.

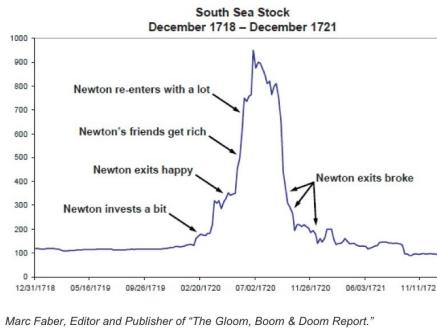


Figure 7.7: South Sea Bubble

In the midst of this frenzy, Parliament passed the Bubble Act in June 1720, ironically just as South Sea stock reached its peak. While ostensibly designed to protect investors from fraudulent schemes, the Act was actually lobbied for by South Sea Company directors seeking to eliminate competition from rival ventures attracting investor capital. The Act required all joint-stock companies to obtain expensive royal charters and imposed severe penalties on unauthorized companies, effectively giving the South Sea Company a monopoly on investor enthusiasm. Paradoxically, by forcing the shutdown of smaller speculative ventures and drying up alternative investments, the Act may have hastened the bubble's collapse by causing investors to question the sustainability of the broader market euphoria. The Act's unintended consequences proved profound and long-lasting—it severely restricted the development of joint-stock companies in Britain for over a century until its repeal in 1825, arguably impeding industrialization by creating legal obstacles for large-scale ventures requiring significant capital. The episode offers enduring lessons about financial markets: price bubbles exhibit the characteristics of non-stationary stochastic processes with time-varying volatility and jump risk, leverage amplifies both gains and losses, regulatory interventions can create unintended consequences, and even rational agents can behave irrationally when caught in speculative manias—all phenomena that modern stochastic models attempt to capture.

The plot of the prices reveals the interconnected nature of early 18th-century financial manias and demonstrates the stochastic features that modern models attempt to capture. The South Sea Bubble exhibits the classic pattern of explosive growth followed by catastrophic collapse—a dramatic jump discontinuity in September 1720 that cannot be explained by continuous Brownian motion alone. Remarkably, the contagion spread across markets: the Bank of England, Royal African Company, and Old East India Company all show synchronized price movements during 1720, rising in sympathy with the South Sea speculation before experiencing their own sharp corrections. Most striking is the Mississippi Company plot, which tracks John Law's concurrent bubble

in France—it peaked slightly earlier than the South Sea Bubble and collapsed even more precipitously, suggesting that speculative manias can propagate across national borders. The synchronization across these four series illustrates volatility clustering and correlation jumps, phenomena that motivate the stochastic volatility models with correlated jumps discussed later in this chapter. These price paths exhibit all three components of the Lévy-Itô decomposition: drift during the accumulation phase, continuous Brownian fluctuations throughout, and sudden Poisson jumps at the moment of collapse.

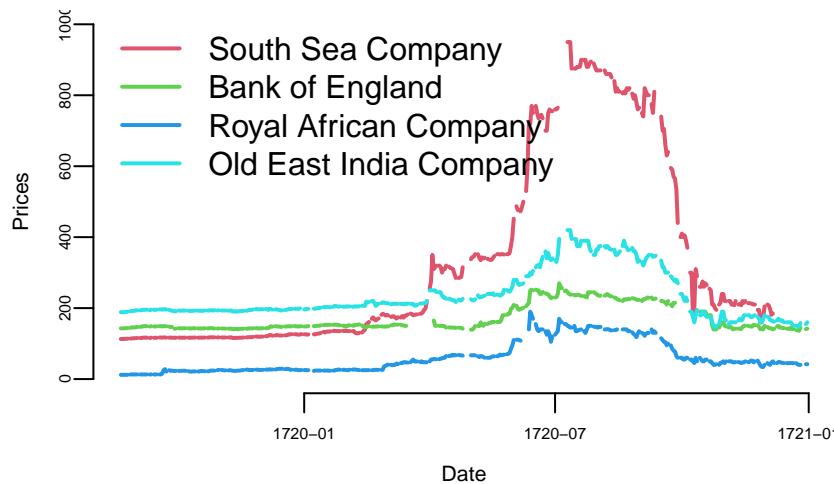


Figure 7.8: Historical Bubbles

7.6 Stochastic Volatility: Financial Economics

Financial markets exhibit time-varying volatility—periods of calm trading alternate with episodes of extreme price movements. The October 1987 crash dramatically illustrated the limitations of constant volatility models: the Dow Jones index fell 23% in a single day, an event that had probability essentially zero under the lognormal model assumed by the Black-Scholes framework. This observation motivated the development of stochastic volatility models that allow uncertainty itself to evolve randomly over time. The crash was precipitated by ‘portfolio insurance,’ an automated strategy that sold equities as markets declined, creating a self-reinforcing downward spiral.

Robert Merton, who was a student of Samuelson, proposed a major extension to the work of Bachelier by introducing jumps to the model. The additive jump term addresses the issues of asymmetry and heavy tails in the distribution. Merton's Jump Stochastic volatility model has a discrete-time version for log-returns, y_t , with jump times, J_t , jump sizes, Z_t , and spot stochastic volatility, V_t , given by the dynamics

$$\begin{aligned} y_t &\equiv \log(S_t/S_{t-1}) = \mu + V_t \varepsilon_t + J_t Z_t \\ V_{t+1} &= \alpha_v + \beta_v V_t + \sigma_v \sqrt{V_t} \varepsilon_t^v \end{aligned}$$

where $\mathbb{P}(J_t = 1) = \lambda$, S_t denotes a stock or asset price and log-returns $y^t = (y_1, \dots, y_t)$ are the log-returns. The errors $(\varepsilon_t, \varepsilon_t^v)$ are possibly correlated bivariate normals. The investor must obtain optimal filters for (V_t, J_t, Z_t) , and learn the posterior densities of the parameters $(\mu, \alpha_v, \beta_v, \sigma_v^2, \lambda)$. These estimates will be conditional on the information available at each time.

7.6.1 Motivation: Combining Brownian Motion and Jumps

The Lévy-Itô decomposition provides the theoretical foundation for modeling asset prices. Recall that any Lévy process can be decomposed into three fundamental components: deterministic drift, continuous Brownian fluctuations, and discrete jumps. In finance, this decomposition maps naturally to observed price dynamics:

- *Drift (μt)*: The expected return on the asset, reflecting long-term growth trends
- *Brownian component (σB_t)*: Continuous price fluctuations driven by the steady arrival of market information
- *Jump component ($\sum_{i=1}^{N_t} Z_i$)*: Sudden price movements triggered by earnings announcements, regulatory changes, or macroeconomic shocks

Traditional models like Black-Scholes use only the first two components, assuming constant volatility σ . However, empirical evidence overwhelmingly shows that volatility itself is stochastic and exhibits its own patterns: it clusters (high volatility follows high volatility), it mean-reverts to long-run averages, and it can experience sudden jumps during crises.

Stochastic volatility models extend the Lévy-Itô framework by allowing the volatility parameter to follow its own stochastic process. The most general formulation combines:

1. **Brownian motion** for continuous price and volatility fluctuations

2. **Poisson processes** for rare but important jump events in both prices and volatility
3. **Correlation structure** to capture the *leverage effect*—the empirical observation that volatility tends to rise when prices fall

This integration of the two fundamental stochastic processes creates a flexible modeling framework capable of capturing the rich dynamics observed in financial markets.

Example 7.6 (Financial Crashes and the Need for Jumps). Consider the distribution of daily S&P 500 returns. Under a Gaussian model with annualized volatility of 15%, a one-day drop of 5% should occur roughly once every 10,000 years. Yet such events occurred multiple times in recent decades: October 1987 (-20.5%), October 2008 (-9.0%), March 2020 (-12.0%). The empirical distribution exhibits *heavy tails*—extreme events occur far more frequently than predicted by the normal distribution.

Jump-diffusion models accommodate these events naturally. Instead of treating crashes as impossible outliers, they model them as rare but expected occurrences from the Poisson jump component. This provides more realistic risk assessment and option pricing, particularly for out-of-the-money puts that protect against market crashes.

7.6.2 The Stochastic Volatility Model

The basic stochastic volatility (SV) model extends the geometric Brownian motion of Black-Scholes by allowing volatility to evolve as a latent stochastic process. In continuous time, the log-price $\log S_t$ and its variance v_t jointly evolve as:

$$\begin{aligned} d \log S_t &= \mu dt + \sqrt{v_t} dB_t^s \\ d \log v_t &= \kappa_v(\theta_v - \log v_t)dt + \sigma_v dB_t^v \end{aligned}$$

where B_t^s and B_t^v are (potentially correlated) Brownian motions. The variance follows a mean-reverting process in logs with:

- θ_v : Long-run average log-variance
- κ_v : Speed of mean reversion (how quickly volatility returns to its average)
- σ_v : Volatility of volatility (how much randomness in the volatility process)

Discretizing this model at daily or weekly intervals yields the discrete-time specification:

$$\begin{aligned}y_t &= \mu + \sqrt{v_{t-1}}\varepsilon_t^s \\ \log v_t &= \alpha_v + \beta_v \log v_{t-1} + \sigma_v \varepsilon_t^v\end{aligned}$$

where $y_t = \log(S_t/S_{t-1})$ are log-returns, $\varepsilon_t^s, \varepsilon_t^v \sim N(0, 1)$ are standard normal innovations, and the parameters relate to the continuous-time specification via $\alpha_v = \kappa_v \theta_v \Delta$ and $\beta_v = 1 - \kappa_v \Delta$ for time interval Δ .

This model exhibits several desirable features:

1. **Volatility clustering:** Since $\log v_t$ follows an AR(1), periods of high volatility tend to persist
2. **Stationarity:** The mean-reverting specification ensures volatility doesn't explode or collapse to zero
3. **Flexibility:** The correlation between ε_t^s and ε_t^v allows for leverage effects

An important empirical regularity in equity markets is that volatility tends to increase when prices fall—a phenomenon known as the *leverage effect*. While originally attributed to changing debt-to-equity ratios as stock prices move, it is now understood as a more general feature of risk dynamics.

To incorporate the leverage effect, we allow the innovations in returns and volatility to be correlated:

$$\begin{aligned}y_t &= \mu + \sqrt{v_{t-1}}\varepsilon_t^s \\ \log v_t &= \alpha_v + \beta_v \log v_{t-1} + \sigma_v [\rho\varepsilon_t^s + \sqrt{1-\rho^2}\varepsilon_t^v]\end{aligned}$$

where $\rho < 0$ for equity returns. A negative return shock ($\varepsilon_t^s < 0$) directly increases log-volatility through the ρ term, generating the observed inverse relationship between prices and volatility.

While stochastic volatility captures the time-varying nature of market uncertainty, it still relies on continuous Brownian motion for price movements. To accommodate the extreme events and heavy tails observed in returns, we augment the model with jump components—invoking the full Lévy-Itô decomposition.

The **stochastic volatility with jumps** (SVJ) model extends the basic SV specification by adding a Poisson-driven jump process to returns:

$$\begin{aligned}y_t &= \mu + \sqrt{v_{t-1}}\varepsilon_t^s + J_t Z_t \\ \log v_t &= \alpha_v + \beta_v \log v_{t-1} + \sigma_v \varepsilon_t^v\end{aligned}$$

where:

- $J_t \sim \text{Bernoulli}(\lambda)$ indicates whether a jump occurs at time t
- $Z_t \sim N(\mu_Z, \sigma_Z^2)$ is the jump size when a jump occurs
- λ is the jump intensity (probability of a jump per period)

The total variance of returns now decomposes into two sources:

$$\text{Var}(y_t) = E(v_{t-1}) + \lambda E[Z_t^2]$$

The first term captures diffusive volatility from continuous fluctuations, while the second captures jump variance from discrete events. This allows the model to simultaneously fit the day-to-day variations (through v_t) and occasional crashes (through jumps).

The 2008 financial crisis revealed another important feature: volatility itself experiences sudden jumps. The VIX index (a measure of market volatility expectations) more than doubled in a matter of days during the Lehman Brothers collapse. To capture this, we extend the model to allow jumps in both returns and volatility.

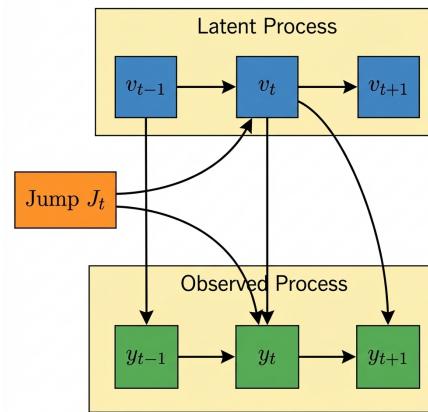


Figure 7.9: Stochastic Volatility Model

The graphical structure of this model is shown in Figure 7.9, illustrating how the latent volatility process v_t evolves over time and influences the observed returns y_t , with jumps J_t affecting both processes simultaneously.

The *stochastic volatility with correlated jumps* (SVCJ) model specifies:

$$\begin{aligned} y_t &= \mu + \sqrt{v_{t-1}} \varepsilon_t^s + J_t Z_t \\ v_t &= \alpha_v + \beta_v v_{t-1} + \sigma_v \sqrt{v_{t-1}} \varepsilon_t^v + J_t W_t \end{aligned}$$

where:

- The same Bernoulli J_t triggers jumps in both returns and volatility (correlated jumps)
- $Z_t|W_t \sim N(\mu_Z + \rho_J W_t, \sigma_Z^2)$ allows jump sizes to be correlated
- $W_t \sim \text{Exponential}(\mu_W)$ ensures volatility jumps are positive

The correlation parameter $\rho_J < 0$ captures the empirical finding that large negative return jumps are typically accompanied by large positive volatility jumps. For example, during the March 2020 COVID-19 crash, the S&P 500 fell sharply while the VIX spiked to record levels.

An even more flexible specification, the **stochastic volatility with independent jumps** (SVIJ) model, allows jumps in returns and volatility to occur independently, governed by separate Poisson processes with intensities λ_Z and λ_W . This provides maximum flexibility but requires more data to estimate reliably.

To understand the empirical importance of these model features, consider parameter estimates from S&P 500 daily returns (1980-1999):

Table 7.7: Model features

Feature	SV	SVJ	SVCJ	SVIJ
Stochastic volatility	+	+	+	+
Return jumps	-	+	+	+
Volatility jumps	-	-	+	+
Independent jumps	-	-	-	+

The estimated average annualized volatility across models is remarkably stable (around 15%), closely matching the sample standard deviation of 16%. However, the decomposition of variance sources differs:

- **SV model:** All variation comes from the stochastic volatility component
- **SVJ model:** 85% from stochastic volatility, 15% from return jumps
- **SVCJ model:** 90% from stochastic volatility, 10% from return jumps
- **SVIJ model:** 92% from stochastic volatility, 8% from return jumps

The diminishing role of return jumps as we add volatility jumps reflects an important finding: much of what appears as “jumps in returns” in simpler models is actually driven by *jumps in volatility*. When volatility suddenly spikes, even Brownian motion can generate large price movements that might be misidentified as jumps.

The mean reversion parameter κ_v also varies across specifications. In the SVCJ and SVIJ models, κ_v roughly doubles compared to the SV model, indicating that volatility reverts more quickly when jumps account for sudden large moves. The volatility-of-volatility parameter σ_v correspondingly falls, as jumps handle the extreme variations.

7.6.3 Bayesian Inference for Stochastic Volatility Models

Estimating stochastic volatility models presents a significant challenge: the volatility v_t is never directly observed, appearing as a latent state variable. Classical maximum likelihood approaches require integrating out the entire volatility path, which is computationally intractable for nonlinear models with jumps.

The Bayesian approach via MCMC provides an elegant solution by treating the latent volatilities and jump indicators as parameters to be sampled alongside model parameters. The Clifford-Hammersley theorem structures the algorithm efficiently.

For the basic SV model with parameter vector $\theta = (\alpha_v, \beta_v, \sigma_v^2)$ and latent volatilities $v = (v_1, \dots, v_T)$, the joint posterior factors as:

$$p(\theta, v|y) \propto p(y|v)p(v|\theta)p(\theta)$$

The MCMC algorithm alternates between:

1. **Parameter update:** $p(\theta|v, y)$

- Given volatilities, returns are conditionally normal: $y_t|v_{t-1} \sim N(\mu, v_{t-1})$
- Log-volatilities follow AR(1): $\log v_t | \log v_{t-1} \sim N(\alpha_v + \beta_v \log v_{t-1}, \sigma_v^2)$
- With conjugate priors, conditional posteriors are standard (Normal-Inverse-Gamma)

2. **Volatility update:** $p(v_t|v_{t-1}, v_{t+1}, \theta, y)$

- The conditional posterior for each v_t combines information from:

- The likelihood $p(y_{t+1}|v_t)$ (observed return depends on current volatility)
- The state evolution $p(v_t|v_{t-1})$ (Markov dynamics from previous period)
- The forward evolution $p(v_{t+1}|v_t)$ (Markov dynamics to next period)

- This distribution is non-standard and requires Metropolis-Hastings sampling

For the jump-augmented models (SVJ, SVCJ, SVIJ), we additionally sample:

3. **Jump indicators:** $p(J_t|v, Z, \theta, y)$

- Each $J_t \in \{0, 1\}$ follows a Bernoulli posterior
- Large observed returns increase the probability of $J_t = 1$

4. **Jump sizes:** $p(Z_t|J_t = 1, v, \theta, y)$

- Conditional on a jump occurring, the jump size has a Normal posterior
- The posterior mean balances the jump prior and the size needed to explain the observed return

This modular structure allows us to build up from simpler models (SV) to more complex specifications (SVIJ) by adding components one at a time, reusing the same basic algorithmic building blocks.

Stochastic volatility models with jumps have become standard tools in quantitative finance for several applications:

Option Pricing: The Black-Scholes model systematically misprices options, particularly out-of-the-money puts. The *volatility smile*—the observation that implied volatilities increase for strikes far from the current price—reflects the market’s recognition of jump risk and stochastic volatility. Jump-diffusion models with stochastic volatility can reproduce these patterns, providing more accurate prices and hedging strategies.

Risk Management: Value-at-Risk (VaR) and Expected Shortfall calculations based on constant-volatility Gaussian models dramatically underestimate tail risks. By properly accounting for stochastic volatility and jumps, firms can better quantify their exposure to extreme market movements. During the 2008 crisis, many institutions discovered their VaR models had severely underestimated potential losses.

Portfolio Allocation: The presence of stochastic volatility creates hedging demands even for long-horizon investors. An investor who correctly anticipates that volatility is mean-reverting will reduce equity exposure when volatility is high (because expected returns are temporarily compressed) and increase exposure when volatility is low. This generates countercyclical trading strategies.

Market Timing: The predictable component of volatility can be exploited for tactical asset allocation. Since volatility tends to mean-revert, unusually high volatility signals elevated future returns (as compensation for risk), making it

an opportune time to increase risky asset exposure. Conversely, unusually low volatility may warrant defensive positioning.

The integration of Brownian motion and Poisson processes through stochastic volatility models exemplifies how the Lévy-Itô decomposition provides not just mathematical elegance, but practical power for understanding and managing financial risk in modern markets.



8

Gaussian Processes

“Uncertainty is the only certainty there is, and knowing how to live with insecurity is the only security.” —John Allen Paulos

In traditional regression, we often find a single “best” line that fits the data (e.g., $y = mx + b$). But reality is rarely so simple. What if we want to model not just the trend, but our **uncertainty** about it? What if, instead of committing to a single function, we could consider an *infinite* number of possible functions consistent with our data?

A **Gaussian Process (GP)** allows us to do exactly that. Instead of estimating parameters for a specific function, we define a probability distribution over *all possible functions* that fit the data. It is a powerful, non-parametric tool used extensively in machine learning for tasks ranging from robotic control to geospatial analysis.

Formally, a GP is a collection of random variables, any finite number of which have a joint Gaussian distribution. A finite collection of n points drawn from a GP is completely specified by its n -dimensional mean vector μ and covariance matrix Σ . We assume the process is indexed by a real variable $x \in \mathbb{R}$ (e.g., time or space) and has real-valued outputs. The GP is defined by: 1. **Mean function** $m(x) = \mathbb{E}[f(x)]$: The expected value of the function at point x . 2. **Covariance (Kernel) function** $k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]$: A measure of similarity between values at x and x' .

We denote this as:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')).$$

Intuitively, the kernel function determines the “shape” and “smoothness” of the functions we expect to see. If x and x' are close, $k(x, x')$ should be high, implying $f(x)$ and $f(x')$ are likely similar.

In practice, we often assume a zero mean, $m(x) = 0$, and focus on the covariance kernel. The choice of kernel encodes our prior beliefs about the data. The most common choice is the **Squared Exponential (SE)** kernel (also known as the Radial Basis Function or RBF):

$$k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right)$$

Here, we have two *hyperparameters*:

- σ^2 (Signal Variance): Controls the vertical amplitude of the function.
- l (Length Scale): Controls the horizontal “wiggleness.” A large l implies the function changes slowly (smooth), while a small l allows for rapid variations.

Observe that $k(x, x) = \sigma^2$ and $k(x, x') \rightarrow 0$ as the distance $|x - x'| \rightarrow \infty$.

We can illustrate a GP with a simulated example. First generate a sequence of 100 input points (indices)

and then define the mean function and the covariance function

The covariance function depends only on the distance between two points, not on their absolute values. The squared exponential kernel is infinitely differentiable, which means that the GP is a very smooth function. The squared exponential kernel is also called the radial basis function (RBF) kernel. The covariance matrix is then defined as

```
cov_mat = outer(x, x, sqexpcov)
```

and we can generate a sample from the GP using the `mvrnorm` function from the `MASS` package and plot a sample.

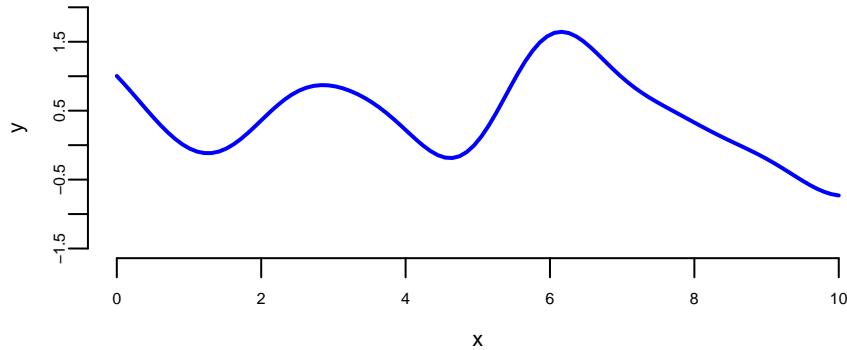


Figure 8.1: Sample from a Gaussian Process

Figure 8.1 displays 100 values of a function $f(x)$ drawn from a GP with zero mean and a squared-exponential kernel at inputs $x = (0, 0.1, 0.2, \dots, 10)$. The realisation is smooth, with most values lying between -2 and 2. Because each diagonal element of the covariance matrix equals $\sigma^2 = 1$, the marginal variance is one. By properties of the normal distribution, approximately 95 percent of the points of Y should therefore fall within 1.96 standard deviations of the

mean. The mild oscillations arise because values with neighbouring indices are highly correlated.

We can generate a few more samples from the same GP and plot them together

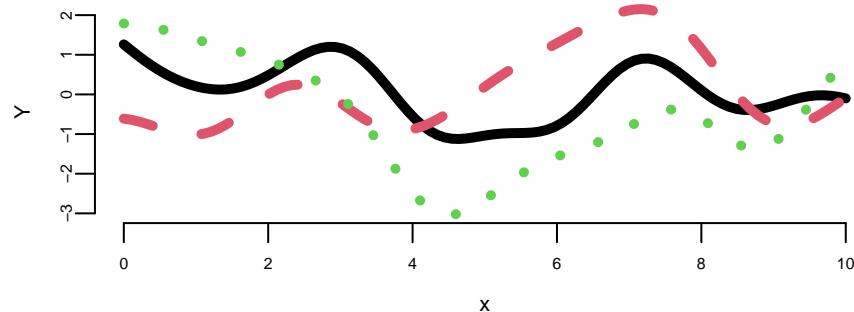


Figure 8.2: Samples from a Gaussian Process

Each finite sample path differs from the next, yet all share a similar range, a comparable number of bumps, and overall smoothness. That's what it means to have function realizations under a GP prior: $Y = f(x) \sim \mathcal{GP}(0, k(x, x'))$

Simulating from the prior shows us the richness of possible functions we can model. However, our goal is not just to generate random curves, but to learn. We want to constrain these possibilities using actual observations.

8.1 Making Predictions with Gaussian Processes

Suppose our observed data consists of n inputs $\mathbf{X} = (x_1, \dots, x_n)^T$ and outputs $\mathbf{y} = (y_1, \dots, y_n)^T$. We assume these are a realization of a GP. Our goal is to predict the outputs \mathbf{y}_* at new inputs \mathbf{X}_* .

By definition of a GP, the joint distribution of the observed data \mathbf{y} and the predictions \mathbf{y}_* is a multivariate Gaussian:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix} \right)$$

where: * $\mathbf{K} = k(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{n \times n}$ is the covariance of the training data. * $\mathbf{K}_* = k(\mathbf{X}, \mathbf{X}_*) \in \mathbb{R}^{n \times q}$ is the covariance between training and test data. * $\mathbf{K}_{**} = k(\mathbf{X}_*, \mathbf{X}_*) \in \mathbb{R}^{q \times q}$ is the covariance of the test data. * $\mu = \mathbb{E}[\mathbf{y}]$ and $\mu_* = \mathbb{E}[\mathbf{y}_*]$.

The beauty of Gaussians is that conditioning on observed data is closed-form. The conditional distribution of \mathbf{y}_* given \mathbf{y} is:

$$\mathbf{y}_* \mid \mathbf{y}, \mathbf{X}, \mathbf{X}_* \sim \mathcal{N}(\mu_{\text{post}}, \Sigma_{\text{post}})$$

The posterior mean μ_{post} serves as our prediction, and the posterior covariance Σ_{post} quantifies our uncertainty:

$$\mu_{\text{post}} = \mu_* + \mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{y} - \mu) \quad (8.1)$$

$$\Sigma_{\text{post}} = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_* \quad (8.2)$$

Equation 8.1 and Equation 8.2 are standard properties of the multivariate normal distribution (see Chapter 3 and Appendix Chapter 26). Intuitively, the posterior variance Σ_{post} is equal to the prior variance \mathbf{K}_{**} minus a term representing the information gained from the observed data. This structure reflects a fundamental Bayesian principle: data acts to reduce our prior uncertainty (represented by the second term in Equation 8.2).

Example 8.1 (Gaussian Process for sin function). We can use the GP to make predictions about the output values at new inputs x_* . We use x in the $[0, 2\pi]$ range and y to be the $y = \sin(x)$. We start by simulating the observed x - y pairs.

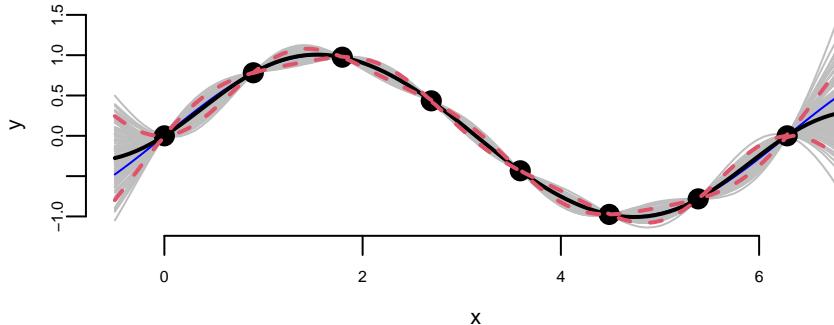
The additive term `diag(eps, n)` corresponds to adding $\epsilon \mathbf{I}$, which stabilizes the matrix inversion by ensuring strict positive definiteness; in machine learning practice, this is known as ‘jitter’. Now we implement a function that calculates the mean and covariance of the posterior distribution of y_* given Y .

Now we generate a new set of inputs x_* and calculate the covariance matrices K_* and K_{**} .

Notice that we did not add ϵI to $K_* = K\mathbf{X}$ matrix, but we add it to $K_{**} = K\mathbf{X}\mathbf{X}^T$ to ensure the posterior covariance is invertible. We do not add it to K_* ($K\mathbf{X}$) as it represents cross-covariance, which does not need to be positive definite. Now we can calculate the mean and covariance of the posterior distribution of y_* given Y .

Now, we can generate a sample from the posterior distribution over y_* , given Y

Using our convenience function `plot_gp` we can plot the posterior distribution over y_* , given Y .

Figure 8.3: Posterior distribution over y_* , given Y

Example 8.2 (Gaussian Process for Simulated Data using MLE). In the previous example, we assumed fixed values for the hyperparameters: $\sigma^2 = 1$ and $l^2 = 0.5$ (since $2l^2 = 1$). In real applications, we don't know these values; we must estimate them from the data.

We use Maximum Likelihood Estimation (MLE) to find the parameters that maximize the probability of observing our data. This parallels the likelihood-to-loss framing in Chapter 11: we write down a (marginal) likelihood for the data and then optimize it, often via its log-likelihood. This section relies on basic matrix operations (inverse, determinant); Appendix Chapter 26 provides a short refresher. If you have not seen gradient-based optimization, the core intuition appears in Chapter 20; here we use it only as a practical tool to fit GP hyperparameters.

The marginal likelihood of the data \mathbf{y} (integrating out the function values f) is:

$$p(\mathbf{y} | \mathbf{X}, \sigma, l) = \frac{1}{(2\pi)^{n/2} |\mathbf{K}|^{1/2}} \exp\left(-\frac{1}{2}\mathbf{y}^T \mathbf{K}^{-1} \mathbf{y}\right)$$

where \mathbf{K} is the covariance matrix computed with hyperparameters σ and l . For numerical stability, we typically maximize the *log*-likelihood:

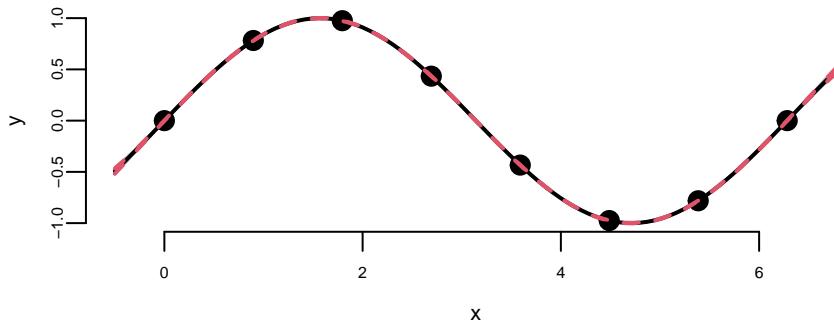
$$\log p(\mathbf{y} | \mathbf{X}, \sigma, l) = -\frac{1}{2} \log |\mathbf{K}| - \frac{1}{2} \mathbf{y}^T \mathbf{K}^{-1} \mathbf{y} - \frac{n}{2} \log 2\pi.$$

This equation encapsulates **Occam's Razor**. The term $-\frac{1}{2}\mathbf{y}^T \mathbf{K}^{-1} \mathbf{y}$ rewards the model for fitting the data well. The term $-\frac{1}{2} \log |\mathbf{K}|$ penalizes model complexity; a more flexible kernel (e.g., smaller length scale) leads to a simpler determinant term that exacts a cost. MLE automatically balances these two competing objectives to prevent overfitting.

We can implement a function that calculates the log-likelihood of the data given the hyperparameters σ and l and use `optim` function to find the maximum of the log-likelihood function.

```
## 1.5 2.4
```

The `optim` function returns the hyperparameters that maximize the log-likelihood function. We can now use those hyperparameters to make predictions about the output values at new inputs x_* .



We can see that our uncertainty is much narrower—the posterior distribution is considerably tighter. This is because we used the observed data to estimate the hyperparameters. We can also see that the posterior mean is closer to the true function $y = \sin(x)$. Although our initial guess of $\sigma^2 = 1$ and $2l^2 = 1$ was not too far off, the model fits the data much better when we use the estimated hyperparameters.

The default `optim` function uses numerical approximations for derivatives. While convenient, this can be slow and less precise. For GPs, we can calculate the analytical gradients of the log-likelihood with respect to the hyperparameters, significantly speeding up optimization.

Mathematical Details: Gradient Derivation

To optimize efficiently, we rely on two matrix calculus identities:

$$\frac{\partial \mathbf{K}^{-1}}{\partial \theta} = -\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta} \mathbf{K}^{-1} \quad \text{and} \quad \frac{\partial \log |\mathbf{K}|}{\partial \theta} = \text{tr} \left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta} \right)$$

Using these, the gradient of the log-likelihood is:

$$\frac{\partial \log p(\mathbf{y} | \mathbf{X}, \theta)}{\partial \theta} = -\frac{1}{2} \text{tr} \left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta} \right) + \frac{1}{2} \mathbf{y}^T \mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta} \mathbf{K}^{-1} \mathbf{y}$$

In the case of squared exponential kernel, the elements of the covariance matrix K are given by

$$K_{ij} = k(x_i, x_j) = \sigma^2 \exp\left(-\frac{1}{2} \frac{(x_i - x_j)^2}{l^2}\right).$$

The derivative of the covariance matrix with respect to σ is given by

$$\frac{\partial K_{ij}}{\partial \sigma} = 2\sigma \exp\left(-\frac{1}{2} \frac{(x_i - x_j)^2}{l^2}\right); \quad \frac{\partial K}{\partial \sigma} = \frac{2}{\sigma} K.$$

The derivative of the covariance matrix with respect to l is given by

$$\frac{\partial K_{ij}}{\partial l} = \sigma^2 \exp\left(-\frac{1}{2} \frac{(x_i - x_j)^2}{l^2}\right) \frac{(x_i - x_j)^2}{l^3}; \quad \frac{\partial K}{\partial l} = \frac{(x_i - x_j)^2}{l^3} K.$$

Now we can implement a function that calculates the derivative of the log-likelihood function with respect to σ and l .

Now we can use the `optim` function to find the maximum of the log-likelihood function and provide the derivative function we just implemented.

```
par1 = optim(c(1,1), fn=loglik, gr=gnlg ,X=X,
             Y=Y,method="BFGS")$par
l = par1[2]; sigma = par1[1]
print(par1)
## 1.5 2.4
```

The result is the same compared to when we called `optim` without the derivative function. Even execution time is the same for our small problem. However, at larger scale, the derivative-based optimization algorithm will be much faster.

Furthermore, instead of coding our own derivative functions, we can use an existing package, such as the `laGP` package, developed by Bobby Gramacy to estimate the hyperparameters. The `laGP` package uses the same optimization algorithm we used above, but it also provides better selection of the covariance functions and implements approximate GP inference algorithms for large scale problems, when n becomes large and inversion of the covariance matrix K is prohibitively expensive.

```
library(laGP)
gp = newGP(X, Y, 1, 0, dK = TRUE)
res = mleGP(gp, tmax=20)
```

```
1.laGP = sqrt(res$d/2)
print(1.laGP)
## 2.4
```

In the `newGP` function defines a Gaussian process with square exponential covariance function and assumes $\sigma^2 = 1$, then `mleGP` function uses optimization algorithm to maximize the log-likelihood and returns the estimated hyperparameters $d = 2l^2$, we can see that the length scale is close to the one we estimated above. We will use the `predplot` convenience function to calculate the predictions and plot the data vs fit.

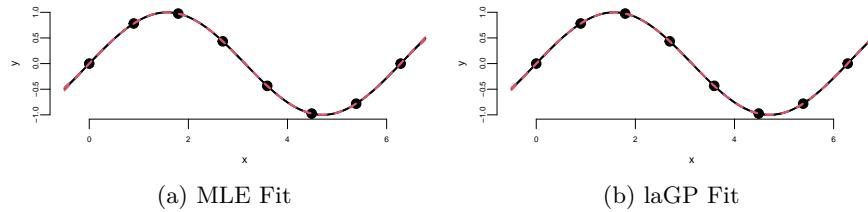
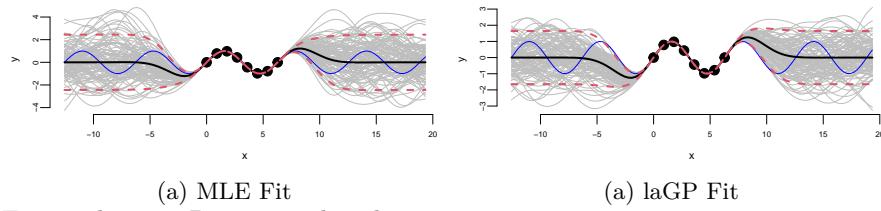


Figure 8.4: Posterior distribution over y_* , given Y

We can see that there is visually no difference between the two fits. Thus, it seems irrelevant whether we keep sigma fixed $\sigma = 1$ or estimate it using MLE. However, in other applications when uncertainty is larger, the choice of σ is important when we use GP for regression and classification tasks. Even for our example, if we ask our model to extrapolate



Extrapolation: Posterior distribution over y_* , given Y

We can see that outside of the range of the observed data, the model with $\sigma = 1$ is more confident in its predictions.

Now, instead of using GP to fit a known function (\sin), we will apply it to a real-world data set. We will use the motorcycle accident data set from the MASS package. The data set contains accelerator readings taken through time

in a simulated experiment on the efficacy of crash helmets. This data is non-linear and exhibits varying curvature, making it an excellent candidate for GP regression where linear models would fail.

Example 8.3 (Gaussian Process for Motorcycle Accident Data). We first estimate the length scale parameter l using the `1aGP` package.

Now we plot the data and the fit using the estimated length scale parameter l .

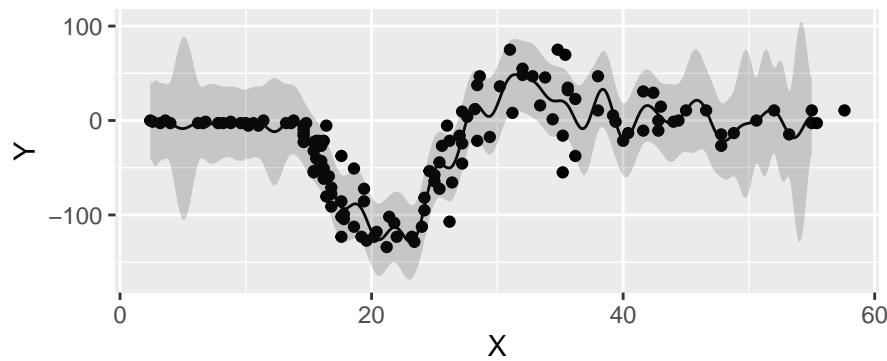


Figure 8.7: Motorcycle Accident Data. Black line is the mean of the posterior distribution over y_* , given Y . Blue lines are the 95% confidence interval.

We can see that our model is more confident for time values between 10 and 30. The confidence interval is wider for time values between 0 and 10 and between 30 and 60, and less confident at the end close to the 60 mark. For some reason the acceleration values were not measured evenly. If we look at the histogram of time values, we can see that there are more data points in the middle of the time range.

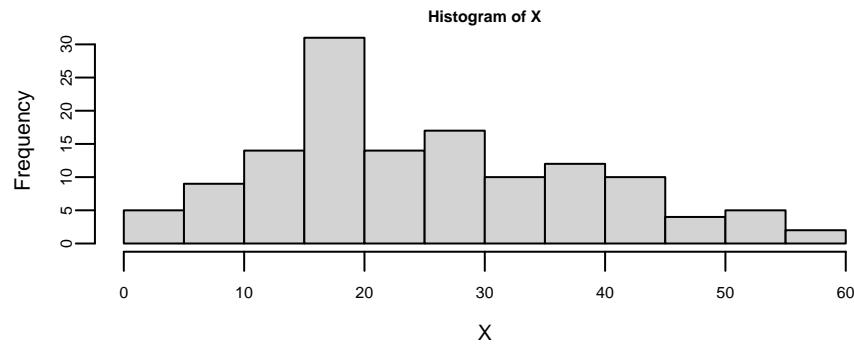


Figure 8.8: Histogram of time values

The widening of the confidence intervals in regions with fewer data points (e.g., between 30 and 40) is a natural property of the GP; where data is sparse, the model reverts to the prior covariance, resulting in higher uncertainty.

In summary, Gaussian Processes provide a robust and flexible framework for modeling functions where uncertainty is key. By defining a prior over functions and updating it with data, we obtain a posterior distribution that captures both predictions and the confidence in those predictions. The key features of GPs are:

- *Non-parametric*: GPs can model functions of arbitrary complexity without a fixed number of parameters.
- *Data-efficient*: They work well with small datasets and provide uncertainty estimates that are crucial for decision-making.
- *Versatile*: Through the choice of the kernel function, GPs can capture various structures (smoothness, periodicity, etc.) and are used in fields ranging from environmental modeling to hyperparameter optimization in deep learning (“Bayesian Optimization”).

9

Reinforcement Learning

“*Information is the resolution of uncertainty.*” Claude Shannon, 1948

Thus far we have discussed making decisions under uncertainty (Chapter 4 and Chapter 5) and two modes of data collection: field experiments and observational studies. In a field experiment we control the data-generation process before the study begins, whereas in an observational study we have no such control and must work with whatever data are produced.

What if we can choose which data to collect *while* the experiment is running? This leads to sequential (or adaptive) experimental design, in which each new observation is selected on the basis of the data gathered so far, creating a feedback loop between data generation and decision-making. The idea is illustrated in the following diagram:

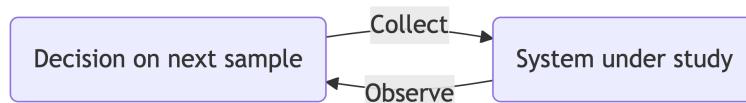


Figure 9.1: Sequential experimental design

This framework supports a wide range of applications and is implemented through several key algorithms. In this section, we will consider the most widely used among them: (i) Multi-Armed Bandits, (ii) Q-Learning, (iii) Active Learning, and (iv) Bayesian Optimization.

One of the first practical demonstrations of reinforcement learning (then called *trial-and-error learning*) was Claude Shannon’s 1950s mechanical mouse *Theseus*, which learned to find its way through a maze.

9.1 Multi-Armed Bandits

When the number of alternatives is large and the testing budget limited, a different approach to A/B testing—the *multi-armed bandit* (MAB)—can be more sample-efficient. MABs allow us to balance *exploration* (trying new things) and *exploitation* (sticking with what works) simultaneously, minimizing *regret* (the difference between our choice and the optimal one). MABs require that the outcome of each experiment is available immediately or with a small delay to decide on the next experiment (Steven L. Scott 2015). When time is of the essence and there is an opportunity cost associated with delaying the decision, MABs are a better choice than traditional A/B testing.

The mathematical framework, the foundational Bayesian solution—Thompson sampling—and a comparison with classical A/B testing are covered in Chapter 6 (see Section 6.5.5). Here we focus on practical aspects: when to end experiments, how to handle contextual information, and design considerations.

Formally, there are K alternatives (arms) and each arm a is associated with a reward distribution v_a , the value of this arm. The goal is to find the arm with the highest expected reward and accumulate the highest total reward in doing so. The reward distribution is unknown and we can only observe the reward after we select an arm a , but we assume that we know the distribution $f_a(y | \theta)$, where a is the arm index, y is the reward, and θ is the set of unknown parameters to be learned. Here are a few examples:

1. In online advertisements, we have K possible ads to be shown and probability of user clicking is given by vector $\theta = (\theta_1, \dots, \theta_K)$ of success probabilities for K independent binomial models, with $v_a(\theta) = \theta_a$. The goal is to find the ad with the highest click-through rate.
2. In website design, we have two design variables, the color of a button (red or blue) and its pixel size (27 or 40), we introduce two dummy variables x_c for color, x_s for size and the probability of user clicking is given by

$$\text{logit}P(\text{click} | \theta) = \theta_0 + \theta_x x_c + \theta_s x_s + \theta_{cs} x_c x_s,$$

with $v_a(\theta) = P(\text{click} | \theta)$.

Variations include controlling for background variables (covariates not under our control, such as time of day or user location) or replacing the binary outcome with a continuous variable (time spent on the website, money spent) using linear regression or another appropriate generalized linear model.

The key challenge is that acting greedily—always choosing the arm with the highest estimated reward $\hat{a} = \arg \max_a v_a(\hat{\theta})$ —may cause us to miss better alternatives. We are not sure about our estimates $\hat{\theta}$ and need to explore other options. Thompson sampling, the oldest and most elegant Bayesian solution, addresses this by sampling from the posterior distribution of each arm’s success probability and selecting the arm with the highest sample. This naturally balances exploration and exploitation: arms with high uncertainty have a chance to produce high samples and get selected, while arms with high posterior means are selected frequently. The algorithm achieves optimal regret bounds of $O(\log T)$ (meaning the regret scales logarithmically with time T) over T time steps.

9.1.1 When to End Experiments

Step 2 of the TS algorithm can be replaced by calculating probability of an arm a being the best w_{at} and then choose the arm by sampling from the discrete distribution w_{1t}, \dots, w_{Kt} . The probability of an arm a being the best is given by

$$w_{at} = P(a \text{ is optimal} \mid y^t) = \int P(a \text{ is optimal} \mid \theta)P(\theta \mid y^t)d\theta,$$

where $y^t = (y_1, \dots, y_t)$ is the history of observations up to time t . We can calculate the probabilities w_{at} using Monte Carlo. We can sample $\theta^{(1)}, \dots, \theta^{(G)}$ from the posterior distribution $p(\theta \mid y^t)$ and calculate the probability as

$$w_{at} \approx \frac{1}{G} \sum_{g=1}^G I(a = \arg \max_i v_i(\theta^{(g)})),$$

where $I(\cdot)$ is the indicator function. This is simply the proportion of times the arm was the best in the G samples.

Although using a single draw from posterior $p(\theta \mid y^t)$ (as in the original algorithm) is equivalent to sampling proportional to w_{at} , the explicitly calculated w_{at} yields a useful statistic that can be used to decide on when to end the experiment.

We will use the regret statistic to decide when to stop. Regret is the difference in values between the truly optimal arm and the arm that is apparently optimal at time t . Although we cannot know the regret (it is unobservable), we can compute samples from its posterior distribution. We simulate the posterior distribution of the regret by sampling $\theta^{(1)}, \dots, \theta^{(G)}$ from the posterior distribution $p(\theta \mid y^t)$ and calculating the regret as

$$r^{(g)} = (v_a^*(\theta^{(g)}) - v_{a_t^*}(\theta^{(g)})),$$

Here a^* is the arm deemed best across all Monte Carlo draws and the first term is the value of the best arm within draw g . We choose a_t^* as

$$a_t^* = \arg \max_a w_{at}.$$

Often, it is convenient to measure the regret on the percent scale and then we use

$$r^{(g)} \leftarrow r^{(g)}/v_{a_t^*}(\theta^{(g)})$$

We can demonstrate using simulated data. The function below generates samples $\theta^{(g)}$

```
bandit <- function(x, n, alpha = 1, beta = 1, ndraws = 5000) {
  set.seed(17) # Kharlamov
  K <- length(x) # number of bandits
  prob <- matrix(nrow = ndraws, ncol = K)
  no <- n - x
  for (a in 1:K) { # posterior draws for each arm
    prob[, a] <- rbeta(ndraws, x[a] + alpha, no[a] + beta)
  }
  prob
}
```

Say we have three arms with 20, 30, and 40 sessions that have generated 12, 20, and 30 conversions. We assume a uniform prior for each arm $\theta_i \sim Beta(1, 1)$ and generate 6 samples from the posterior of $\theta | y^t$.

```
x <- c(12, 20, 30)
n <- c(20, 30, 40)
prob <- bandit(x, n, ndraws = 6)
```

	θ_1	θ_2	θ_3
1	0.60	0.63	0.58
2	0.62	0.62	0.74
3	0.69	0.53	0.67
4	0.49	0.59	0.73
5	0.61	0.51	0.69
6	0.47	0.64	0.69

Now, we calculate the posterior probabilities $w_{at} = P(a \text{ is optimal} \mid y^t)$ for each of the three arms

```
wat <- table(factor(max.col(prob), levels = 1:3)) / 6
```

1	2	3
0.17	0.17	0.67

Thus far, the third arm is the most likely to be optimal, with probability 67%. Now, we calculate the regret for each of the six draws from the posterior of $\theta | y^t$.

```
regret <- (apply(prob, 1, max) - prob[, 3]) / prob[, 3]
```

1	2	3	4	5	6
0.09	0	0.03	0	0	0

We compute value row by row by subtracting the largest element of that row from the element in column 3 (because arm 3 has the highest chance of being the optimal arm). All rows but 1 and 3 are zero. In the first row, the value is $(0.63-0.58)/0.58$ because column 2 is 0.05 larger than column 3. If we keep going down each row we get a distribution of values that we could plot in a histogram. We can generate one for a larger number of draws (10000).

```
prob <- bandit(x, n, ndraws = 10000)
regret <- (apply(prob, 1, max) - prob[, 3]) / prob[, 3]
```

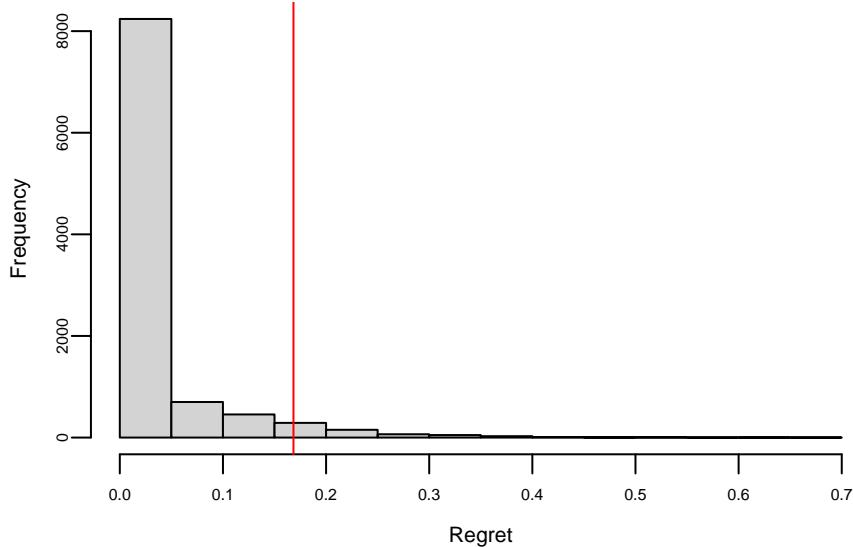


Figure 9.2: The histogram of the value remaining in an experiment (regret). The vertical line is the 95th percentile, or the potential value remaining.

```
wat <- table(factor(max.col(prob), levels = 1:3)) / 10000
```

	1	2	3
	0.08	0.2	0.72

The histogram of the value remaining in an experiment (regret). The vertical line is the 95th percentile, or the potential value remaining.

Arm 3 has a 72% probability of being the best arm, so the value of switching away from arm 3 is zero in 72% of the cases. The 95th percentile of the value distribution is the potential value remaining (CvR) in the experiment, which in this case works out to be about 16%.

```
quantile(regret, 0.95)
## 95%
## 0.17
```

You interpret this number as “We’re still unsure about the CvR for arm 3, but whatever it is, one of the other arms might beat it by as much as 16%.”

Google Analytics, for example, “ends the experiment when there’s at least a 95% probability that the value remaining in the experiment is less than 1% of the champion’s conversion rate. That’s a 1% improvement, not a one percentage point improvement. So if the best arm has a conversion rate of 4%, then we end the experiment if the value remaining in the experiment is less than .04 percentage points of CvR”.

9.1.2 Contextual Bandits

Traditional multi-armed bandit models, like the binomial model, assume independent observations with fixed reward probabilities. This works well when rewards are consistent across different groups and times. However, for situations with diverse user bases or fluctuating activity patterns, such as international audiences or browsing behavior, this assumption can be misleading.

For instance, companies with a global web presence may experience temporal effects as markets in Asia, Europe, and the Americas become active at different times of the day. Additionally, user behavior can change based on the day of the week, with people engaging in different browsing patterns and purchase behaviors. For example, individuals may research expensive purchases during work hours but make actual purchases on weekends.

Consider an experiment with two arms, A and B. Arm A performs slightly better during the weekdays when users browse but don’t make purchases, while Arm B excels during the weekends when users are more likely to make purchases. High traffic volume might lead a binomial model to declare Arm A the winner before observing any weekend behavior. This risk exists regardless of whether the experiment is conducted as a bandit or a traditional experiment. Bandit experiments, however, are particularly vulnerable to this bias due to their typically shorter durations.

To mitigate the risk of being misled by distinct sub-populations, two methods can be employed. If the specific sub-populations are known in advance or if there is a proxy for them, such as geographically induced temporal patterns, the binomial model can be adapted to a logistic regression model. This modification allows for a more nuanced understanding of the impact of different factors on arm performance, helping to account for variations in sub-population behavior and temporal effects.

$$\text{logit}P(\text{click}_a \mid \theta, x) = \theta_{0a} + \beta^T x,$$

where x describes the circumstances or context of the observation. The success probability for selecting arm a under the context x is represented as $P(\text{click}_a \mid \theta, x)$. Each arm a has its own specific coefficient denoted as β_{0a} with one arm’s coefficient set to zero as a reference point. Additionally, there is another set of coefficients represented as β that are associated with the contextual data

and are learned as part of the model. The value function can then be

$$v_a(\theta) = \text{logit}^{-1}(\beta_{0a}).$$

If we lack knowledge about the crucial contexts, one option is to make the assumption that contexts are generated randomly from a context distribution. This approach is often exemplified by the use of a hierarchical model like the beta-binomial model.

$$\begin{aligned}\theta_{at} &\sim \text{Beta}(\alpha_a, \beta_a) \\ \text{click}_a \mid \theta &\sim \text{Binomial}(\theta_{at}),\end{aligned}$$

where $\theta = \{\alpha_a, \beta_a : a = 1, \dots, K\}$, with value function $v_a(\theta) = \alpha_a / \beta_a$

9.1.3 Summary of MAB Experimentation

The design phase begins with defining your arms by identifying the different options you want to evaluate, such as different website layouts, pricing strategies, or marketing campaigns. Next, choose a bandit algorithm that balances exploration and exploitation in various ways. Popular choices include Epsilon-greedy, Thompson Sampling, and Upper Confidence Bound (UCB). Then set your parameters by configuring the algorithm parameters based on your priorities and expected uncertainty. For example, a higher exploration rate encourages trying new arms earlier. Finally, randomize allocation by assigning users to arms randomly, ensuring unbiased data collection.

During the analysis phase, track rewards by defining and measuring the reward metric for each arm, such as clicks, conversions, or profit. Monitor performance by regularly analyzing the cumulative reward and arm selection probabilities to see which arms are performing well and how the allocation strategy is adapting. Use statistical tools like confidence intervals or Bayesian methods to compare performance between arms and assess the significance of findings. Make adaptive adjustments by modifying the experiment based on ongoing analysis. You might adjust algorithm parameters, stop arms with demonstrably poor performance, or introduce new arms.

Start with a small pool of arms to avoid information overload by testing a manageable number of options initially. Set a clear stopping criterion by deciding when to end the experiment based on a predetermined budget, time limit, or desired level of confidence in the results. Consider ethical considerations by ensuring user privacy and informed consent if the experiment involves personal data or user experience changes. Interpret results in context by remembering that MAB results are specific to the tested conditions and might not generalize perfectly to other contexts.

By following these steps and utilizing available resources, you can design and analyze effective MAB experiments to optimize your decision-making in various scenarios. Remember to adapt your approach based on your specific goals and context to maximize the benefits of this powerful technique.

9.2 Bellman Principle of Optimality

“An optimum policy has the property that whatever the initial state and initial decision are, the remaining decision sequence must be optimum for the state resulting from the first decision.” – Richard Bellman

To solve sequential decision problems like the one above, we rely on a fundamental concept in dynamic programming.

Example 9.1 (Secretary Problem). The Secretary Problem, also known as the marriage problem or sultan’s dowry problem, is a classic problem in decision theory and probability theory. The scenario involves making a decision on selecting the best option from a sequence of candidates or options. The problem is often framed as hiring a secretary, but it can be applied to various situations such as choosing a house, a spouse, or any other scenario where you sequentially evaluate options and must make a decision.

In this problem, you receive T offers and must either accept or reject the offer “on the spot”. You cannot return to a previous offer once you have moved on to the next one. Offers are in random order and can be ranked against those previously seen. The aim is to maximize the probability of choosing the offer with the greatest rank. There is an optimal r ($1 \leq r < T$) to be determined such that we examine and reject the first r offers. Then of the remaining $T - r$ offers we choose the first one that is best seen to date.

A decision strategy involves setting a threshold such that the first candidate above this threshold is hired, and all candidates below the threshold are rejected. The optimal strategy, known as the 37% rule, suggests that one should reject the first $r = T/e$ candidates and then select the first candidate who is better than all those seen so far.

The reasoning behind the 37% rule is based on the idea of balancing exploration and exploitation. By rejecting the first T/e candidates, you gain a sense of the quality of the candidates but avoid committing too early. After that point, you select the first candidate who is better than the best among the initial r candidates.

It’s important to note that the 37% rule provides a probabilistic guarantee

of selecting the best candidate with a probability close to $1/e$ (approximately 37%) as T becomes large.

To solve the secretary problem, we will use the principle of optimality due to Richard Bellman. The principle states that an optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. In other words, the policy is optimal from the first decision onwards.

The solution to the secretary problem can be found via dynamic programming. Given an agent with utility function $u(x, d)$, with current state x , and decision d . The law of motion of x_t is given by $x_{t+1} = p(x_t, d_t)$. Bellman principle of optimality states that the optimal policy is given by the following recursion

$$V(x_t) = \max_{d_t} \{u(x_t, d_t) + \gamma E(V(x_{t+1}))\}$$

where γ is the discount factor. The optimal policy is given by

$$d_t^* = \arg \max_{d_t} \{u(x_t, d_t) + \gamma E(V(x_{t+1}))\}.$$

Now, back to the secretary problem. Let $y^t = (y_1, \dots, y_t)$ denote the history of observations up to time t . State $x_t = 1$ if the t th candidate is the best seen so far and $x_t = 0$ otherwise. The decision $d_t = 1$ if the t th candidate is hired and $d_t = 0$ otherwise. The utility function is given by $u(x_t, d_t) = x_t d_t$. The Bellman equation is given by

$$P(\text{best of T} \mid x_t = 1) = \frac{P(\text{best of T})}{P(x_t = 1)} = \frac{1/T}{1/t} = \frac{t}{T}.$$

The t th offer is the best seen so far places no restriction on the relative ranks of the first $t - 1$ offers. Therefore,

$$p(x_t = 1, y^{t-1}) = p(x_t = 1)p(y^{t-1})$$

by the independence assumption. Hence, we have

$$p(x_t = 1 \mid y^{t-1}) = p(x_t = 1) = \frac{1}{t}.$$

Let $p^*(x_{t-1} = 0)$ be the probability under the optimal strategy. Now we have to select the best candidate, given we have seen $t - 1$ offers so far and the last one was not the best or worse. The probability satisfies the Bellman equation

$$p^*(x_{t-1} = 0) = \frac{t-1}{t} p^*(x_t = 0) + \frac{1}{t} \max(t/T, p^*(x_t = 0)).$$

This leads to

$$p^*(x_{t-1} = 0) = \frac{t-1}{T} \sum_{\tau=t-1}^{T-1} \frac{1}{\tau}.$$

Remember, the strategy is to reject the first r candidates and then select the first. The probability of selecting the best candidate is given by

$$P(\text{success}) = \frac{1}{T} \sum_{a=r+1}^T \frac{r}{a} \approx \frac{1}{T} \int_r^T \frac{r}{a} da = \frac{r}{T} \log\left(\frac{T}{r}\right).$$

We optimize over r by setting the derivative to zero:

$$\frac{\log\left(\frac{T}{r}\right)}{T} - \frac{1}{T} = 0,$$

which gives the optimal $r = T/e$.

If we plug in $r = T/e$ back to the probability of success, we get

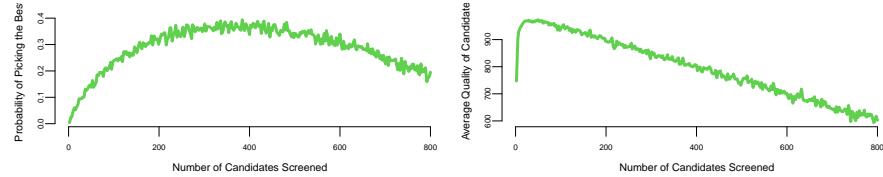
$$P(\text{success}) \approx \frac{1}{e} \log(e) = \frac{1}{e}.$$

Monte Carlo Simulations Simulations are a powerful tool for making decisions when we deal with a complex system, which is difficult or impossible to analyze mathematically. They are used in many fields, including finance, economics, and engineering. They can also be used to test hypotheses about how a system works and to generate data for statistical analysis.

We start by showing how the secretary problem can be analyzed using simulations rather than analytical derivations provided above.

```
plot(d$rules, d$cnt / d$nmc,
  type = "l", col = 3, lwd = 3, xlab = "Number of Candidates
  ↪ Screened",
  ylab = "Probability of Picking the Best"
)
plot(d$rules, d$quality / 1000,
  type = "l", col = 3, lwd = 3, xlab = "Number of Candidates
  ↪ Screened",
  ylab = "Average Quality of Candidate"
)
```

The left panel confirms the theoretical result: the probability of selecting the best candidate peaks at approximately 37% when we screen roughly 370 out of 1000 candidates (close to $T/e \approx 368$). The right panel reveals an interesting trade-off: average quality peaks much earlier, around 50–100 candidates



screened, then steadily declines. This happens because being too selective (screening too many) increases the risk of rejecting all strong candidates and being forced to accept the last one, regardless of quality. The two plots together illustrate the tension between *maximizing the chance of finding the absolute best* (left) and *maximizing expected quality* (right)—objectives that lead to different optimal stopping rules.

9.3 Markov Decision Processes

A Markov Decision Process (MDP) is a discrete time stochastic control process which provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. Almost all dynamic programming and reinforcement learning problems are formulated using the formalism of MDPs. MDPs were known at least as early as the 1950s; a core body of research resulted from Ronald Howard's 1960 book, [Dynamic Programming and Markov Processes](#). In fact, the multi-armed bandit problem considered before is a special case of MDP with one state.

An MDP is defined by:

1. *States (S):* A set of states representing different scenarios or configurations. A key assumption is the Markov property: the future depends only on the current state and action, not on the history.
2. *Actions (A):* A set of actions available in each state.
3. *Transition Probability (P):* $P(s', r | s, a)$ is the probability of transitioning to state s' , receiving reward r , given that action a is taken in state s .
4. *Reward (R):* A reward function $R(s, a, s')$ that gives the feedback signal immediately after transitioning from state s to state s' , due to action a .
5. *Discount Factor (γ):* A factor between 0 and 1, which reduces the value of future rewards.

9.3.1 Mathematical Representation

The states s_t and rewards R_t in MDP are indexed by time t . The state at time $t + 1$ is distributed according to the transition probability

$$P(s_{t+1} | s_t, a_t).$$

The reward function is $R_s^a = \mathbb{E}(R_{t+1} | s, a)$.

The Markov property of the state is that the transition probability depends only on the current state and action and not on the history of states and actions.

$$P(s_{t+1} | s_t, a_t) = P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0).$$

In other words, the future only depends on the present and not on the past history. The state is a sufficient statistic for the future.

In the case when the number of states is finite, we can represent the transition probability as a matrix $P_{ss'}^a = P(s_{t+1} = s' | s_t = s, a_t = a)$, where $s, s' \in S$ and $a \in A$. For a given action a , the transition probability matrix P^a is a square matrix of size $|S| \times |S|$, where each row sums to 1

$$P^a = \begin{bmatrix} P_{11}^a & P_{12}^a & \cdots & P_{1|S|}^a \\ P_{21}^a & P_{22}^a & \cdots & P_{2|S|}^a \\ \vdots & \vdots & \ddots & \vdots \\ P_{|S|1}^a & P_{|S|2}^a & \cdots & P_{|S||S|}^a \end{bmatrix}$$

The reward function is also a matrix $R_s^a = E[R_{t+1} | s_t = s, a_t = a]$.

Markov Reward Process

We can consider a simpler example of Markov Process. This is a special case of MDP when there is no action and the transition probability is simply a matrix $P_{ss'} = P(s_{t+1} = s' | s_t = s)$, where $s, s' \in S$. For a given action a , the transition probability matrix P^a is a square matrix of size $|S| \times |S|$, where each row sums to 1.

Example 9.2 (Student Example). The graph below represents possible states (nodes) and transitions (links). Each node has reward assigned to it which corresponds to the reward function $R(s)$. The transition probabilities are shown on the links. The graph represents a Markov Chain where transitions are probabilistic and not controlled by an agent.

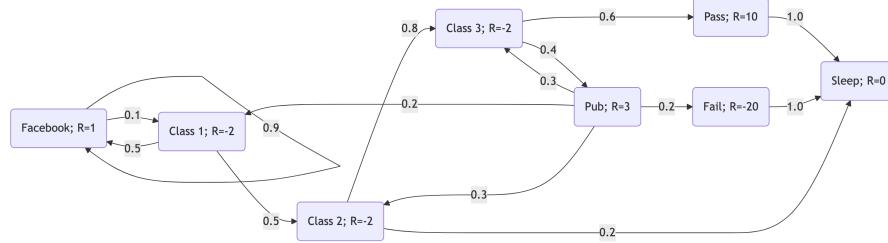


Figure 9.3: Student Example

If we are to pick an initial state and sample a trajectory (path on the graph above) by picking a random action at each state, we will get a random walk on the graph. The reward for each state is shown in the graph. The discounted value of the trajectory is then

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where γ is the discount factor. The discount factor is a number between 0 and 1 that determines the present value of future rewards. A discount factor of 0 makes the agent myopic and only concerned about immediate rewards. A discount factor of 1 makes the agent strive for a long-term high reward. The discount factor is usually denoted by γ and is a parameter of the MDP. The discount of less than 1 is used to avoid infinite returns in cyclic Markov chains and allows us to discount less certain future rewards. The value of γ is usually close to 1, for example 0.9 or 0.99. The value of γ can be interpreted as the probability of the agent surviving from one time step to the next.

We can calculate sample returns G_t for this Markov Chain. We first read in the reward matrix

Table 9.5: Rewards

	Facebook	Class 1	Class 2	Class 3	Pub	Pass	Fail	Sleep
Reward	-1	-2	-2	-2	3	10	-20	0

and the transition probability matrix and the reward matrix

	Facebook	Class.1	Class.2	Class.3	Pub	Pass	Fail	Sleep
Facebook	.9	.1
Class 1	.5	.	.5
Class 282

	Facebook	Class.1	Class.2	Class.3	Pub	Pass	Fail	Sleep
Class 34	.6	.	.
Pub	.	.2	.3	.3	.	.	.2	.
Pass	1.
Fail	1.	.
Sleep	1.

Now we check that all of the rows sum to 1

Table 9.7: Transition probability matrix row sums

Facebook	Class 1	Class 2	Class 3	Pub	Pass	Fail	Sleep
1	1	1	1	1	1	1	1

Given the transition probability matrix, we can sample possible trajectories. First, we define a `tr(s,m)` convenience function that generates a trajectory of length m starting from state s

Now, we generate 6 trajectories of length 5 starting from state “Pub”

Pub	Class 3	Pub	Class 2	Class 3	Pass
Pub	Class 2	Class 3	Pass	Sleep	Sleep
Pub	Class 2	Class 3	Pub	Fail	Fail
Pub	Fail	Fail	Fail	Fail	Fail
Pub	Fail	Fail	Fail	Fail	Fail
Pub	Class 3	Pass	Sleep	Sleep	Sleep

Now we can calculate the discounted value G_t of each of the trajectories

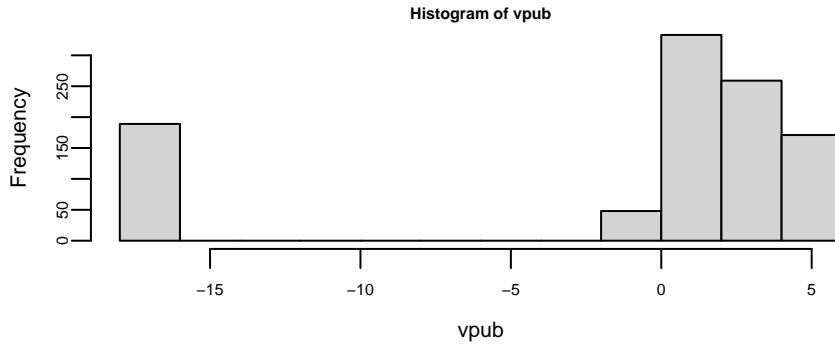
Table 9.9: Discounted value of each trajectory

2.7	2.8	0	-16	-16	4.5
-----	-----	---	-----	-----	-----

We can calculate the discounted value for 1000 trajectories

```
# Value function of a trajectory
value <- function(s, m, gamma = 0.5) {
  traj <- tr(s, m)
  disc <- gamma^(0:m)
  return(sum(sapply(traj, getR) * disc))
```

```
}
vpub <- replicate(1000, value("Pub", 6))
hist(vpub)
```



```
mean(vpub)
## -1.2
```

We can see that the distribution of discounted rewards is bimodal and depends on whether you get to “Fail” state or not.

The value of a state is the expected discounted reward starting from that state

$$V(s) = \mathbb{E}(G_t | s_t = s).$$

It evaluates the long-term value of state s (the goodness of a state). It can be drastically different from the reward value associated with the state. In our student example, the reward for the “Pub” state is 3, but the value is -1.2.

The value of a state can be calculated recursively using the Bellman equation

$$\begin{aligned} V(s) &= \mathbb{E}(G_t | s_t = s) \\ &= \mathbb{E}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t = s) \\ &= \mathbb{E}(R_{t+1} + \gamma G_{t+1} | s_t = s) \\ &= \mathbb{E}(R_{t+1} + \gamma V(s_{t+1}) | s_t = s) \\ &= \sum_{s'} P(s' | s) [R(s) + \gamma V(s')]. \end{aligned}$$

Example 9.3 (MDP for Forest Management). We can consider one of the classic examples of a Markov Decision Process (MDP). Imagine you need to calculate an optimal policy to manage a forest to prevent possible fires. The goal is to decide between two possible actions to either ‘Wait’ or ‘Cut’. They

correspond to balancing between ecological preservation and economic gain, considering the random event of a fire. We can break down the elements of this model.

1. States: Represent the age of the forest. The states are denoted as $\{0, 1, \dots, S - 1\}$, where 0 is the youngest state (just after a fire or cutting) and $S - 1$ is the oldest state of the forest.

2. Actions: There are two actions available:

- ‘Wait’ (Action 0): Do nothing and let the forest grow for another year.
- ‘Cut’ (Action 1): Harvest the forest, which brings immediate economic benefit but resets its state to the youngest.

3. Probabilities: There’s a probability ‘ p ’ each year that a fire occurs, regardless of the action taken. If a fire occurs, the forest returns to state 0.

4. Transition Matrix (P): This matrix defines the probabilities of moving from one state to another, given a specific action.

We will use `mdp_example_forest` function from the MDPtoolbox package to generate the transition probability matrix and reward matrix for the Forest example.

This function generates a transition probability P of size $(|A| \times |S| \times |S|)$. There are four states by default $S = \{1, 2, 3, 4\}$ and two actions $A = \{1, 2\}$. The transition probability matrices for each action are:

$$P(\cdot | s, a = 1) = \begin{bmatrix} 0.01 & 0.99 & 0.00 & 0.00 \\ 0.01 & 0.00 & 0.99 & 0.00 \\ 0.01 & 0.00 & 0.00 & 0.99 \\ 0.01 & 0.00 & 0.00 & 0.99 \end{bmatrix}, \quad P(\cdot | s, a = 2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The reward matrix R of size $|S| \times |A|$ specifies the immediate reward for taking each action in each state:

$$R = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 10 & 1 \end{bmatrix}$$

With these transition probabilities and rewards defined, we can solve for the optimal policy using either value iteration or policy iteration. Value iteration works by iteratively updating the value function until convergence, while policy iteration alternates between policy evaluation and policy improvement

steps. Both algorithms converge to the same optimal policy, though they may differ in computational efficiency depending on the problem structure. For this simple MDP with a discount factor of $\gamma = 0.9$, both methods quickly converge to the optimal policy $\pi^* = (1, 2, 2, 1)$, meaning action 1 is optimal in states 1 and 4, while action 2 is optimal in states 2 and 3. This policy maximizes the expected discounted cumulative reward.

Example 9.4 (Game of Chess as an MDP). We can consider a simple example of a game of chess.

In chess, a state s represents the configuration of the chessboard at any given time. This includes the positions of all the pieces (pawns, knights, bishops, rooks, queen, and king) for both players (white and black). Each possible arrangement of these pieces on the chessboard is a unique state. The game starts in a standard initial state (the standard chess setup) and progresses through a series of states as moves are made. If the game is played to completion, it ends in a terminal state (checkmate, stalemate, or draw). Additionally, the game may be declared a draw if the same board state occurs multiple times, requiring the state representation to track repetition history. Thus, we need to remember the states we have seen before and essentially expand the state space to include the number of times we have seen the state. In a timed game, the game can also end when a player runs out of time.

Actions a in chess are the legal moves that can be made by the player whose turn it is to move. This includes moving pieces according to their allowed movements, capturing opponent pieces, and special moves like castling or en passant. The set of actions available changes with each state, depending on the position of the pieces on the board.

In chess, the transition probability is deterministic for the most part, meaning that the outcome of a specific action (move) is certain and leads to a predictable next state. For example, moving a knight from one position to another (assuming it's a legal move) will always result in the same new configuration of the chessboard. However, in the context of playing against an opponent, there is uncertainty in predicting the opponent's response, which can be seen as introducing a probabilistic element in the larger view of the game.

Defining a reward function R in chess can be complex. In the simplest form, the reward could be associated with the game's outcome: a win, loss, or draw. Wins could have positive rewards, losses negative, and draws could be neutral or have a small positive/negative value. Alternatively, more sophisticated reward functions can be designed to encourage certain strategies or positions, like controlling the center of the board, protecting the king, or capturing opponent pieces.

Chess is a game of perfect information, meaning all information about the game state is always available to both players. While the number of states

in chess is finite, it is extremely large, making exhaustive state analysis (like traditional MDP methods) computationally impractical.

In practice, solving chess as an MDP, especially using traditional methods like value iteration or policy iteration, is not feasible due to the enormous state space. Modern approaches involve heuristic methods, machine learning, and deep learning techniques. For instance, advanced chess engines and AI systems like AlphaZero use deep neural networks and reinforcement learning to evaluate board positions and determine optimal moves, but they do not solve the MDP in the classical sense.

The goal in an MDP is to find a policy $a = \pi(s)$ (a function from states to actions) that maximizes the sum of discounted rewards:

$$V^\pi(s) = \mathbb{E}_\pi(G_t | S_t = s),$$

where

$$G_t = \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})$$

Function $V^\pi(s)$ is the value of state s under policy π . Similarly we can define the action-value function $Q^\pi(s, a)$ as the value of taking action a in state s under policy π :

$$Q^\pi(s, a) = \mathbb{E}_\pi(G_t | S_t = s, A_t = a).$$

Bellman Equations for MDP simply state that the value of a state is the sum of the immediate reward and the discounted value of the next state

$$V^\pi(s) = \mathbb{E}_\pi(R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s) = \sum_{a \in A} \pi(a | s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^\pi(s') \right).$$

The action-value function satisfies the following Bellman equation

$$Q^\pi(s, a) = \mathbb{E}_\pi(R_{t+1} + \gamma Q^\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a).$$

The value function can be defined as an expectation over the action-value function

$$V^\pi(s) = \mathbb{E}_\pi(Q^\pi(s, a) | S_t = s) = \sum_{a \in A} \pi(a | s) Q^\pi(s, a).$$

In matrix form, we have

$$Q^\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^\pi(s') = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' | s') Q^\pi(s', a').$$

Now we can define the Bellman equation in the matrix form

$$V^\pi = R^\pi + \gamma P^\pi V^\pi.$$

The direct solution is then

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi.$$

The optimal value function $V^*(s)$ is the value function for the optimal policy $\pi^*(s)$

$$V^*(s) = \max_\pi V^\pi(s).$$

The optimal action-value function $Q^*(s, a)$ is the action-value function for the optimal policy $\pi^*(s)$

$$Q^*(s, a) = \max_\pi Q^\pi(s, a).$$

The optimal policy $\pi^*(s)$ is the policy that maximizes the value function

$$\pi^*(s) = \arg \max_a Q^*(s, a).$$

The optimal value function satisfies the Bellman optimality equation

$$V^*(s) = \max_a Q^*(s, a).$$

and vice-versa

$$Q^*(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^*(s').$$

The Bellman optimality equation is non-linear and is typically solved iteratively using Value Iteration, Policy Iteration, or Q-learning. Q-learning is an off-policy algorithm that learns the optimal policy by directly estimating the optimal action-value function $Q^*(s, a)$. The algorithm iteratively updates the action-value function using the Bellman optimality backup. The off-policy means that the algorithm learns the optimal policy while following a different policy. The algorithm can learn the optimal policy while following a random policy, for example.

Example 9.5 (Q-Learning and Deal or No Deal). Deal or No Deal is a popular TV show where a contestant is presented with a number of sealed boxes, each containing a prize. The contestant selects a box and then proceeds to open the remaining boxes one by one. After a certain number of boxes have been opened, the banker makes an offer to buy the contestant's box. The contestant can either accept the offer and sell the box or reject the offer and continue opening boxes. The game continues until the contestant either accepts an offer or opens all the boxes. The goal is to maximize the expected value of the prize in the contestant's box. The rule of thumb is to continue as long as there are two large prizes left. Continuation value is large. For example, with three prizes and two large ones, risk averse people will naively choose deal, when if they incorporated the continuation value they would choose no deal.

Let s denote the current state of the system and a an action. The Q -value, $Q_t(s, a)$, is the value of using action a today and then proceeding optimally in

the future. We use $a = 1$ to mean no deal and $a = 0$ means deal. The Bellman equation for Q -values becomes

$$Q_t(s, a) = u(s, a) + \sum_{s^*} P(s^*|s, a) \max_a Q_{t+1}(s^*, a)$$

where $u(s, a)$ is the immediate utility of taking action a in state s . The value function and optimal action are given by

$$V(s) = \max_a Q(s, a) \quad \text{and} \quad a^* = \operatorname{argmax}_a Q(s, a)$$

Transition Matrix: Consider the problem where you have three prizes left. Now s is the current state of three prizes.

$$s^* = \{\text{all sets of two prizes}\} \quad \text{and} \quad P(s^*|s, a = 1) = \frac{1}{3}$$

where the transition matrix is uniform to the next state. There's no continuation for $P(s^*|s, a = 0)$.

Utility: The utility of the next state depends on the contestant's value for money and the bidding function of the banker

$$u(B(s^*)) = \frac{B(s^*)^{1-\gamma} - 1}{1 - \gamma}$$

in power utility case.

Expected value implies $B(s) = \bar{s}$ where s are the remaining prizes.

The website uses the following criteria: with three prizes left:

$$B(s) = 0.305 \times \text{big} + 0.5 \times \text{small}$$

and with two prizes left

$$B(s) = 0.355 \times \text{big} + 0.5 \times \text{small}$$

Three prizes left: $s = \{750, 500, 25\}$.

Assume the contestant is risk averse with log-utility $U(x) = \log x$. Banker offers the expected value we get

$$u(B(s = \{750, 500, 25\})) = \log(1275/3) = 6.052$$

and so $Q_t(s, a = 0) = 6.052$.

In the continuation problem, $s^* = \{s_1^*, s_2^*, s_3^*\}$ where $s_1^* = \{750, 500\}$ and $s_2^* = \{750, 25\}$ and $s_3^* = \{500, 25\}$.

We'll have offers 625, 387.5, 137.5 under the expected value. As the banker offers expected value the optimal action at time $t+1$ is to take the deal $a = 0$ with Q-values given by

$$\begin{aligned} Q_t(s, a = 1) &= \sum_{s^*} P(s^*|s, a = 1) \max_a Q_{t+1}(s^*, a) \\ &= \frac{1}{3} (\log(625) + \log(387.5) + \log(262.5)) = 5.989 \end{aligned}$$

as immediate utility $u(s, a) = 0$. Hence as

$$Q_t(s, a = 1) = 5.989 < 6.052 = Q_t(s, a = 0)$$

the optimal action is $a^* = 0$, deal. Continuation value is not large enough to overcome the generous (expected value) offered by the banker.

Sensitivity analysis: we perform it by assuming different Banker's bidding function. If we use the function from the website (2 prizes):

$$B(s) = 0.355 \times \text{big} + 0.5 \times \text{small},$$

Hence

$$\begin{aligned} B(s_1^* = \{750, 500\}) &= 516.25 \\ B(s_2^* = \{750, 25\}) &= 278.75 \\ B(s_3^* = \{500, 25\}) &= 190 \end{aligned}$$

The optimal action with two prizes left for the contestant is

$$\begin{aligned} Q_{t+1}(s_1^*, a = 1) &= \frac{1}{2} (\log(750) + \log(500)) = 6.415 \\ &> 6.246 = Q_{t+1}(s_1^*, a = 0) = \log(516.25) \\ Q_{t+1}(s_1^*, a = 1) &= \frac{1}{2} (\log(750) + \log(25)) = 4.9194 \\ &< 5.63 = Q_{t+1}(s_1^*, a = 0) = \log(278.75) \\ Q_{t+1}(s_1^*, a = 1) &= \frac{1}{2} (\log(500) + \log(25)) = 4.716 \\ &< 5.247 = Q_{t+1}(s_1^*, a = 0) = (516.25) \end{aligned}$$

Hence future optimal policy will be no deal under s_1^* , and deal under s_2^*, s_3^* .

Therefore solving for Q-values at the previous step gives

$$\begin{aligned} Q_t(s, a = 1) &= \sum_{s^*} P(s^*|s, a = 1) \max_a Q_{t+1}(s^*, a) \\ &= \frac{1}{3} (6.415 + 5.63 + 5.247) = 5.764 \end{aligned}$$

with a monetary equivalent as $\exp(5.764) = 318.62$.

With three prizes we have

$$\begin{aligned} Q_t(s, a = 0) &= u(B(s = \{750, 500, 25\})) \\ &= \log(0.305 \times 750 + 0.5 \times 25) \\ &= \log(241.25) = 5.48. \end{aligned}$$

The contestant is offered \$ 241.

Now we have $Q_t(s, a = 1) = 5.7079 > 5.48 = Q_t(s, a = 0)$ and the optimal action is $a^* = 1$, no deal. The continuation value is large. The premium is \$ 241 compared to \$319, a 33% premium.

9.3.2 MDP Solvers

The underlying approach behind all MDP solvers is to iteratively apply the Bellman equations until convergence. The main difference between the solvers is how they update the value function. All of them use dynamic programming approach to find optimal policy. Dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. It is applicable to problems exhibiting the properties of overlapping subproblems and optimal substructure. If a problem can be solved by combining optimal solutions to non-overlapping subproblems, the strategy is called divide and conquer instead. This is why dynamic programming is applicable to solving MDPs.

First, we consider how to find the values of states under a given policy π . We can iteratively apply Bellman expectation backup. We update the values using the following update rule

$$V_{k+1}(s) = \sum_a \pi(a | s) \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma V_k(s')].$$

We will introduce the short-cut notation

$$P_{ss'}^a = P(s' | s, a), \quad R_s^a = \sum_{s'} P(s' | s, a) R(s, a, s').$$

Then in matrix form the update rule becomes

$$V_{k+1} = R^\pi + \gamma P^\pi V_k.$$

Policy Iteration

The policy iteration algorithm involves two main steps: policy evaluation and policy improvement, which are iteratively applied until convergence. We start

with an arbitrary value function, often initialized to zero for all states.

$$\begin{aligned} V_0(s) &= 0 \\ V_{k+1} &= R^\pi + \gamma P^\pi V_k \\ \pi_{k+1} &= \arg \max_a R^a + \gamma P^a V_{k+1} = \arg \max_a Q^\pi(s, a) \end{aligned}$$

The last step is to simply choose the action that maximizes the expected return in each state. Although it can be slow in practice, the convergence is guaranteed because the value function is a contraction mapping. We typically stop the iterations when the maximum change in the value function is below a threshold.

It can be used for calculating the optimal policy. The Bellman optimality equation is a fundamental part of finding the best policy in MDPs.

$$V^*(s) = \max_a \sum_{s', r} P(s', r | s, a) [r + \gamma V^*(s')]$$

The optimal policy is then

$$\pi^*(s) = \arg \max_a \sum_{s', r} P(s', r | s, a) [r + \gamma V^*(s')]$$

The optimal policy is the one that maximizes the value function. The optimal value function is the value function for the optimal policy. The optimal value function satisfies the Bellman optimality equation. The optimal policy can be found by maximizing the right hand side of the Bellman optimality equation.

Given an optimal policy, we can subdivide it into two parts: the optimal first action A^* and the optimal policy from the next state S' . The optimal value V^* can be found using one-step lookahead

$$V^*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^*(s')$$

Value Iteration This allows us to define another approach to solving MDPs, called value iteration. The value iteration algorithm starts with an arbitrary value function and iteratively applies the Bellman optimality backup. The algorithm updates the value function using the following update rule

$$V_{k+1}(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s').$$

In matrix form, the update rule becomes

$$V_{k+1} = \max_a R^a + \gamma P^a V_k.$$

The algorithm stops when the maximum change in the value function is below a threshold. The optimal policy can be found by maximizing the right hand side of the Bellman optimality equation

$$\pi^*(s) = \arg \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^*(s').$$

In practice, exactly solving the Bellman Expectation Equation in the policy evaluation step can be computationally expensive for large state spaces. Approximate methods may be used. Policy Iteration is particularly effective when the optimal policy needs to be very precise, as in high-stakes decision-making environments.

Example 9.6 (MDP for a Maze). We use a [mazemdp](#) archive by Sally Gao, Duncan Rule, Yi Hao to demonstrate the value and policy iterations. Both are applied to the problem of finding an optimal policy for a maze. The maze is represented as a grid, with each cell either being a wall or empty. Agent (decision maker) does not know the maze structure and needs to find the optimal path from the start to the goal state. The agent starts in the bottom right corner and needs to reach the top left corner (marked as red). The agent can move up, down, left, or right, but not diagonally (actions). Moving into a wall keeps the agent in the same cell. Reaching the goal state gives a reward of +1, and all other transitions give a reward of 0. The goal is to find the optimal policy that maximizes the expected return (sum of discounted rewards) for the agent. In other words, the agent needs to find the shortest path to the exit.

Figures below show the snapshot from policy (top row) and value (bottom row) iterations.

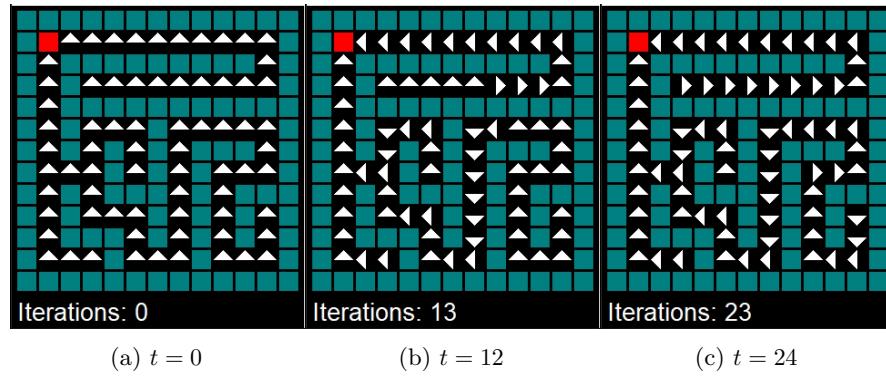


Figure 9.4: Policy Iteration Solver

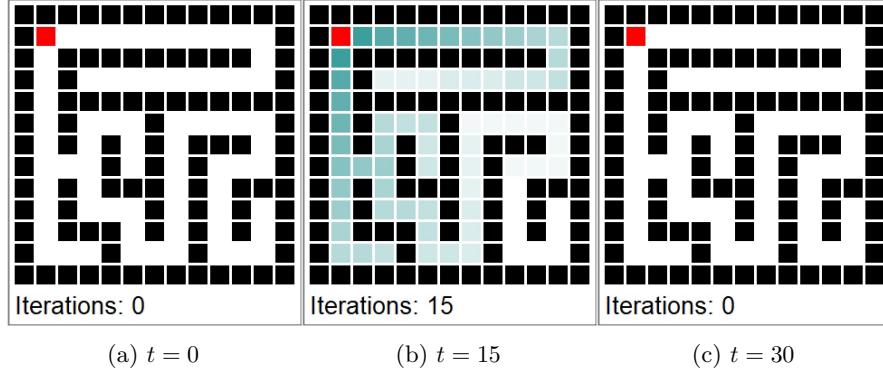


Figure 9.5: Value Iteration Solver

The policy iterations converged after 24 iterations.

A more general form of a value function is the action-value function $Q^\pi(s, a)$, which represents the expected return when starting from state s , taking action a , and following policy π thereafter.

$$Q^\pi(s, a) = \mathbb{E}_\pi(G_t \mid s_t = s, a_t = a).$$

We can derive both the value and optimal policy functions from the action-value function:

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) Q^\pi(s, a)$$

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

9.3.3 Model-Free Methods

Both policy and value iterations we've considered thus far assume that transition probabilities between states given actions are known. However, this is often not the case in many real-world problems. Model-free methods learn through trial and error, by interacting with the environment and observing the rewards. The first method we consider is Monte Carlo methods. Monte Carlo methods for Markov Decision Processes (MDPs) are a class of algorithms used for finding optimal policies when the model of the environment (i.e., the transition probabilities and rewards) is unknown or too complex to model explicitly. These methods rely on learning from experience, specifically from complete episodes of interaction with the environment. Here's a detailed look at how Monte Carlo methods work in the context of MDPs:

1. **Generate Episodes:** An episode is a sequence of states, actions, and rewards, from the start state to a terminal state.

$$S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T \sim \pi.$$

In Monte Carlo methods, these episodes are generated through actual or simulated interaction with the environment, based on a certain policy.

2. **Estimate Value Functions:** Unlike dynamic programming methods, which update value estimates based on other estimated values, Monte Carlo methods update estimates based on actual returns received over complete episodes. This involves averaging the returns received after visits to each state. We use empirical mean to estimate the expected value.
3. **Policy Improvement:** After a sufficient number of episodes have been generated and value functions estimated, the policy is improved based on these value function estimates.

Monte Carlo methods require complete episodes to update value estimates. This makes them particularly suitable for episodic tasks, where interactions naturally break down into separate episodes with clear starting and ending points. MC methods require sufficient exploration of the state space. This can be achieved through various strategies, like ϵ -greedy policies, where there's a small chance of taking a random action instead of the current best-known action. In this case, the policy is given by

$$\pi(a | s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = \arg \max_{a'} Q(s, a') \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases}$$

where ϵ is the probability of taking a random action and $|A|$ is the number of actions. The ϵ -greedy policy is an example of an exploration-exploitation strategy, where the agent explores the environment by taking random actions (exploration) while also exploiting the current knowledge of the environment by taking the best-known action (exploitation). The value of ϵ is typically decayed over time, so that the agent explores more in the beginning and exploits more later on.

Monte Carlo methods are model-free, meaning they do not require a model of the environment (transition probabilities and rewards). They are also effective in dealing with high variance in returns, which can be an issue in some environments. However, they can be inefficient due to high variance and the need for many episodes to achieve accurate value estimates. They also require careful handling of the exploration-exploitation trade-off. The two main approaches for Monte Carlo methods are first-visit and every-visit methods.

1. First-Visit MC: In this approach, the return for a state is averaged over all first visits to that state in each episode.

2. Every-Visit Monte Carlo: Here, the return is averaged over every visit to the state, not just the first visit in each episode.

Monte Carlo Policy Iteration involves alternating between policy evaluation (estimating the value function of the current policy using Monte Carlo methods) and policy improvement (improving the policy based on the current value function estimate). This process is repeated until the policy converges to the optimal policy.

To find the optimal policy, a balance between exploration and exploitation must be maintained. This is achieved through strategies like ϵ -greedy exploration. In Monte Carlo Control, the policy is often improved in a greedy manner based on the current value function estimate.

Recall that an arithmetic average can be updated recursively

$$\bar{x}_n = \frac{1}{n} \sum_{i=1}^n x_i = \frac{1}{n} \left(x_n + \sum_{i=1}^{n-1} x_i \right) = \frac{1}{n} (x_n + (n-1)\bar{x}_{n-1}) = \bar{x}_{n-1} + \frac{1}{n}(x_n - \bar{x}_{n-1}).$$

This is called a running average. We can use this recursion to update the value function $V(s)$ incrementally, each time we visit state s at time t .

$$V(s_t) = V(s_t) + \frac{1}{N(s_t)}(G_t - V(s_t)),$$

where $N(s_t)$ is the number of times we visited state s_t before time t and G_t is the return at time t . This is called first-visit Monte Carlo method. Alternatively, we can use every-visit Monte Carlo method, where we update the value function each time we visit state s .

Alternatively, we can use a learning rate α

$$V_{n+1} = V_n + \alpha(G_n - V_n).$$

This is called constant step size update. The learning rate is a hyperparameter that needs to be tuned. The constant step size update is more convenient because it does not require keeping track of the number of visits to each state. The constant step size update is also more robust to non-stationary problems.

Temporal Difference Learning (TD Learning) Similar to MC, TD methods learn directly from raw experience without a model of the environment. However, unlike MC methods, TD methods update value estimates based on other learned estimates, without waiting for the end of an episode. This is called bootstrapping. TD methods combine the sampling efficiency of Monte Carlo methods with the low variance of dynamic programming methods. They are also model-free and can learn directly from raw experience. However, they are more complex than MC methods and require careful tuning of the learning rate.

A simple TD method is TD(0), which updates value estimates based on the current reward and the estimated value of the next state. The update rule is

$$V(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)),$$

where α is the learning rate. The TD(0) method is also called one-step TD because it only looks one step ahead. The $R_{t+1} + \gamma V(S_{t+1})$ term is called the TD target and is a biased estimate of $V(S_t)$. The difference $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error. The TD target is an estimate of the return G_t and the TD error is the difference between the TD target and the current estimate $V(S_t)$. Although TD algorithms have lower variance than MC methods, they have higher bias. In practice TD methods are more efficient than MC methods.

9.4 Q-Learning

Q-learning is an off-policy algorithm that learns the optimal policy by directly estimating the optimal action-value function $Q^*(s, a)$. The algorithm iteratively updates the action-value function using the Bellman optimality backup. The off-policy means that the algorithm learns the optimal policy while following a different policy. The algorithm can learn the optimal policy while following a random policy, for example. The algorithm can be summarized as follows:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)),$$

where α is the learning rate. The algorithm can be summarized as follows:

1. Initialize $Q(s, a)$ arbitrarily
2. Repeat for each episode:
 1. Initialize S
 2. Repeat for each step of the episode:
 1. Choose A from S using policy derived from Q (e.g., ϵ -greedy)
 2. Take action A , observe R, S'
 3. $Q(S, A) = Q(S, A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S, A))$
 4. $S = S'$
 3. Until S is terminal

Then we can simplify the update rule to

$$Q(S_t, A_t) = (1 - \alpha)Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a)).$$

9.5 Bayesian Optimization

Bayesian Optimization solves an optimization problem using sequential design of experiments, which is the same in spirit as reinforcement learning but focuses on finding the optimum of a static function rather than a policy.

Bayesian optimization is a sequential design strategy for global optimization of black-box functions that does not assume any functional forms. It is particularly useful when the objective function is expensive to evaluate. Bayesian optimization uses a surrogate model to approximate the objective function and an acquisition function to decide where to sample next. The surrogate model is typically a Gaussian process (GP) model, which is a probabilistic model that defines a distribution over functions. The acquisition function is a heuristic that trades off exploration and exploitation to decide where to sample next. Bayesian optimization is a global optimization method, meaning it does not require derivatives and can find the global optimum of the objective function. It is also sample-efficient, meaning it can find the optimum with fewer samples than other methods. However, it can be slow in practice and is not suitable for high-dimensional problems.

Given a function $f(x)$ that is not known analytically (it can represent, for example, output of a complex computer program), the goal is to optimize

$$x^* = \arg \min_x f(x).$$

The Bayesian approach to this problem is the following:

1. Define a prior distribution over $f(x)$
2. Calculate f at a few points x_1, \dots, x_n
3. Repeat until convergence:
 1. Update the prior to get the posterior distribution over $f(x)$
 2. Choose the next point x^+ to evaluate $f(x)$
 3. Calculate $f(x^+)$
4. Pick x^* that corresponds to the smallest value of $f(x)$ among evaluated points

The prior distribution is typically a Gaussian process (GP) model, which is a probabilistic model that defines a distribution over functions. The GP model is defined by a mean function $m(x)$ and a covariance function $k(x, x')$. The mean function is typically set to zero. The covariance function is typically a squared exponential function

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right),$$

where σ_f^2 is the signal variance and l is the length scale. The covariance function defines the similarity between two points x and x' . The covariance function is also called a kernel function. The kernel function is a measure of similarity between inputs x and x' .

Now we need to decide where to sample next. We can use the acquisition function to decide where to sample next. The acquisition function is a heuristic that trades off exploration and exploitation to decide where to sample next. The expected improvement (EI) function is a popular acquisition function. Suppose

$$f^* = \min y$$

is the minimum value of $f(x)$ among evaluated points. At a given point x and function value $y = f(x)$, the expected improvement function is defined as

$$a(x) = E(\max(0, f^* - y)),$$

The function that we calculate expectation of

$$u(x) = \max(0, f^* - y)$$

is the utility function. Thus, the acquisition function is the expected value of the utility function.

The acquisition function is high when y is likely to be lower than f^* , and low when y is likely to be higher than f^* . Given the GP prior, we can calculate the acquisition function analytically. The posterior distribution of Normal $y \sim N(\mu, \sigma^2)$, then the acquisition function is

$$\begin{aligned} a(x) &= E(\max(0, f^* - y)) \\ &= \int_{-\infty}^{\infty} \max(0, f^* - y) \phi(y, \mu, \sigma^2) dy \end{aligned}$$

Since we are interested in improvement $f^* - y > 0$, i.e., $y < f^*$, the integral is from $-\infty$ to f^*

$$= \int_{-\infty}^{f^*} (f^* - y) \phi(y, \mu, \sigma^2) dy$$

where $\phi(y, \mu, \sigma^2)$ is the probability density function of the normal distribution. A useful identity is

$$\int y \phi(y, \mu, \sigma^2) dy = \frac{1}{2} \mu \operatorname{erf}\left(\frac{y - \mu}{\sqrt{2}\sigma}\right) - \frac{\sigma e^{-\frac{(y-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}},$$

where $\Phi(y, \mu, \sigma^2)$ is the cumulative distribution function of the normal distribution. Thus,

$$\int_{-\infty}^{f^*} y \phi(y, \mu, \sigma^2) dy = \frac{1}{2} \mu (1 + \operatorname{erf}\left(\frac{f^* - \mu}{\sqrt{2}\sigma}\right)) - \frac{\sigma e^{-\frac{(f^*-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}} = \mu \Phi(f^*, \mu, \sigma^2) + \sigma^2 \phi(f^*, \mu, \sigma^2).$$

we can write the acquisition function as

$$a(x) = \frac{1}{2} (\sigma^2 \phi(f^*, \mu, \sigma^2) + (f^* - \mu) \Phi(f^*, \mu, \sigma^2))$$

We can implement it

Example 9.7 (Taxi Fleet Optimisation). We will use the taxi fleet simulator from [Emukit project](#). For a given demand (the frequency of trip requests) and the number of taxis in the fleet, it simulates the taxi fleet operations and calculates the profit. The simulator is a black-box function, meaning it does not have an analytical form and can only be evaluated at specific points. The goal is to find the optimal number of taxis in the fleet that maximizes the profit. We will use Bayesian optimization to solve this problem.

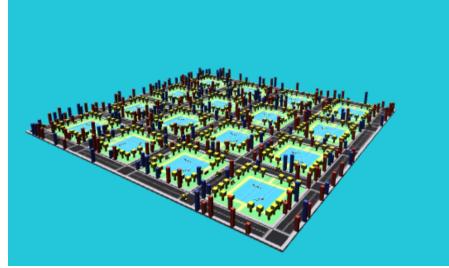


Figure 9.6: Taxi Simulator Visualization

We start with initial set of three designs $x = (10, 30, 90)$, where x is the number of the taxis in the fleet and observe the corresponding profits profit=(3.1,3.6,6.6). When $x = 10$, the demand for taxis exceeds the supply and passengers need to wait for their rides, leading to missed profit opportunities. At another extreme when we have 90 taxis, the profit is slightly better. However, there are many empty taxis, which is not profitable. The optimal number of taxis must be somewhere in the middle. Finally, we try 30 taxis and observe that the profit is higher than both of our previous attempts. However, should we increase or decrease the number of taxis from here? We can use Bayesian optimization to answer this question. First we define a convenience function to plot the GP emulator.

Now, we fit the GP emulator using our initial set of observed taxi-profit pairs, the result is shown in Figure 9.7.

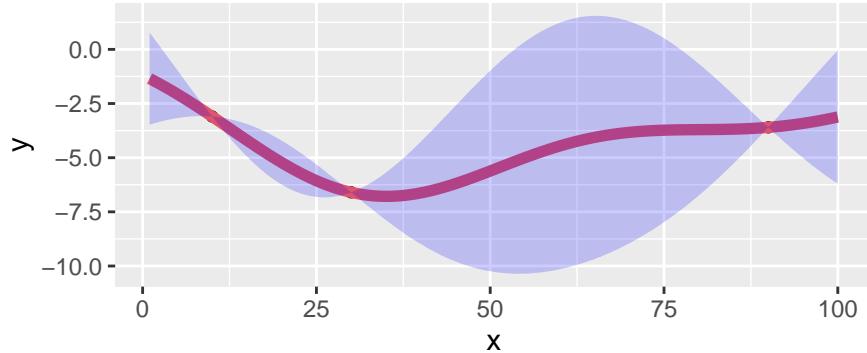


Figure 9.7: Initial GP emulator fit to the taxi-profit pairs.

Instead of maximizing the profit, we minimize the negative profit. We see that there is potentially a better value at around 50 taxis. We can use the acquisition function to decide where to sample next. We define two functions: `nexsample` that uses the acquisition function to decide where to sample next and `updgp` that updates the GP emulator with the new sample. Then we call those two functions twice. First time, EI suggests 44 and second time it suggests 42. We update the GP emulator with the new samples and plot the updated emulator. We see that the GP emulator is updated to reflect the new samples, shown in figure below.

If we run `nexsample` one more time, we get 47, close to our current best of 45. Further, the model is confident at this location. It means that we can stop the algorithm and declare victory, shown in Figure 9.16.

```
## 47
```

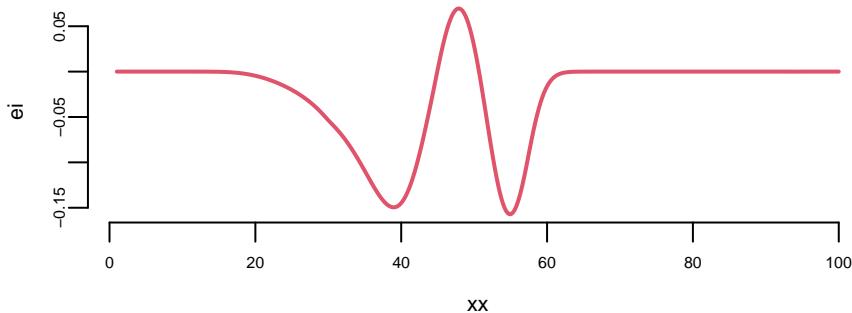
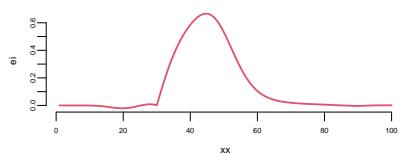
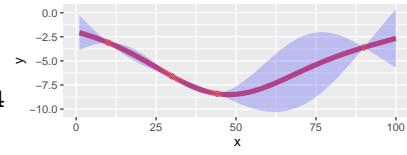


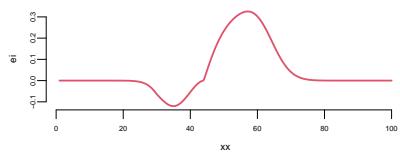
Figure 9.16: The acquisition function after 5 samples. The maximum is at 47.

**## 44**

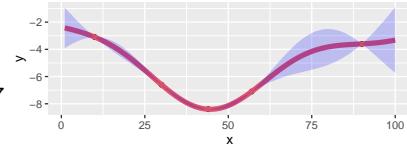
(a) Updated GP emulator fit to the taxi-profit pairs.



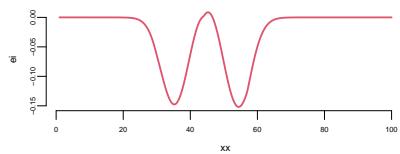
(a) Updated GP emulator fit to the taxi-profit pairs.

**## 57**

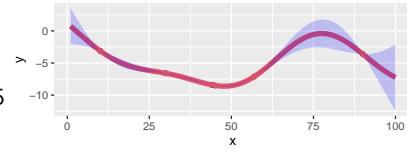
(a) Updated GP emulator fit to the taxi-profit pairs.



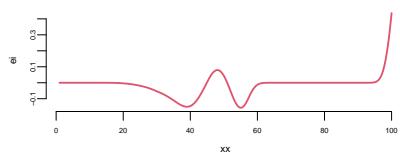
(a) Updated GP emulator fit to the taxi-profit pairs.

**## 45**

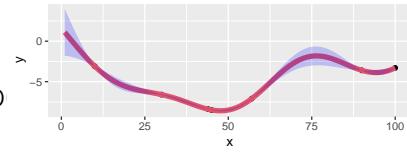
(a) Updated GP emulator fit to the taxi-profit pairs.



(a) Updated GP emulator fit to the taxi-profit pairs.

**## 10**

(a) Updated GP emulator fit to the taxi-profit pairs.



(a) Updated GP emulator fit to the taxi-profit pairs.

9.6 Concluding Remarks

In this chapter, we explored the foundational concepts of Markov Decision Processes (MDPs) and their central role in reinforcement learning. We saw how MDPs provide a flexible mathematical framework for modeling sequential decision-making under uncertainty, with applications ranging from robotics and operations research to online recommendation systems and resource allocation in business.

Through both analytical derivations and Monte Carlo simulations, we examined classic problems such as the secretary problem and taxi fleet optimization, illustrating how simulation and Bayesian optimization can be used to make effective decisions in complex, uncertain environments. The use of Gaussian Process (GP) emulators and acquisition functions like Expected Improvement (EI) demonstrates the power of combining probabilistic modeling with principled exploration strategies—a hallmark of modern reinforcement learning.

As you continue your study of reinforcement learning, remember that the real world rarely presents us with simple, fully known models. The techniques introduced here—modeling uncertainty, simulating outcomes, and iteratively improving decisions—are essential tools for tackling the challenges of real-world AI and data-driven decision making. Whether you are optimizing ad placements, managing supply chains, or designing intelligent agents, the principles of MDPs and Bayesian optimization will serve as a strong foundation for your work.



Part II

AI



10

Unreasonable Effectiveness of Data

It is remarkable that a science which began with the consideration of games of chance should have become the most important object of human knowledge. Pierre Simon Laplace, 1812

Chapters in Part I developed a Bayesian language for uncertainty, learning, and decisions, including sequential decisions. We now shift emphasis from inference about parameters to prediction of future or unseen outcomes. This shift changes what matters: scalability, out-of-sample performance, and how representation choices influence generalization. One useful distinction for Part II is between pattern matching, understood broadly as function approximation, and statistical learning, where the emphasis remains on inference, uncertainty quantification, and the consequences of modeling assumptions. The next chapters use classical statistical models as stepping stones toward modern machine learning, while keeping uncertainty and decision-making as recurring themes rather than afterthoughts.

Data collected by early telescopes played a crucial role in the development of statistical techniques during the 18th century, just as Internet and mobile devices do in the 21st. Massive astronomical data sets inspired scientists such as Carl Friedrich Gauss, Pierre-Simon Laplace, and Simeon Denis Poisson to devise data-driven methods, including the method of least squares and the Poisson distribution. These methods fundamentally transformed science, shifting the focus from purely theoretical derivation to the rigorous, quantitative interrogation of observational data. The integration of data and statistical methods laid the foundation for modern data science and statistics, demonstrating the power and versatility of data-driven approaches.

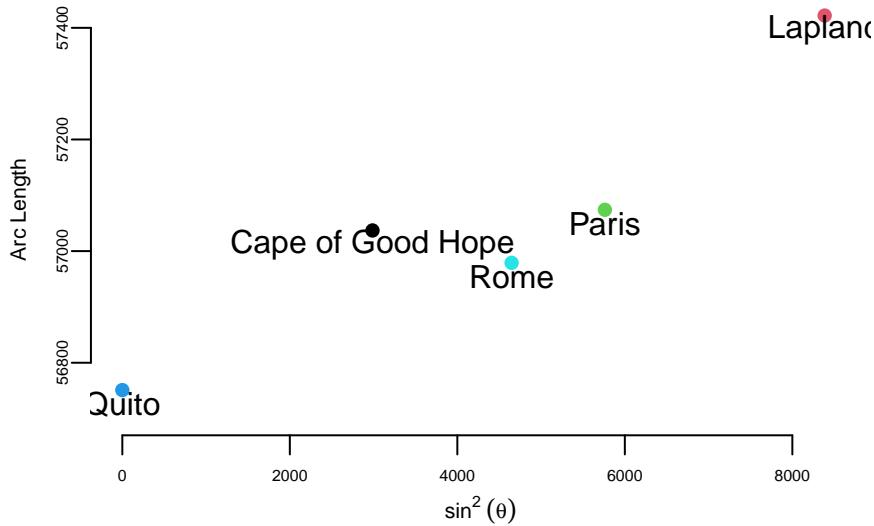
In the 18th and 19th centuries data collection was often limited to manual measurements or observations, and the amount of available data was typically much smaller compared to the massive datasets encountered in modern data science. Scientists like Gauss and Poisson often conducted carefully designed experiments, collected their own data, and performed manual calculations without the aid of computers or advanced statistical software. The focus of their work was often on theoretical developments in mathematics, physics and astronomy, and the data was used to test and validate their theories. We can consider one of those studies from the early 18th century.

Example 10.1 (Boscovich and Shape of Earth). The 18th century witnessed heated debates surrounding the Earth's precise shape. While the oblate spheroid model - flattened poles and bulging equator - held sway, inconsistencies in measurements across diverse regions fueled uncertainty about its exact dimensions. The French, based on extensive survey work by Cassini, maintained the prolate view while the English, based on gravitational theory of Newton (1687), maintained the oblate view.

The determination of the exact figure of the earth would require very accurate measurements of the length of a degree along a single meridian. The final answer to this debate was given by Roger Boscovich (1711-1787) who used geodetic surveying principles and in collaboration with English Jesuit Christopher Maire, in 1755, they embarked on a bold project: measuring a meridian arc spanning a degree of latitude between Rome and Rimini. He employed ingenious techniques to achieve remarkable accuracy for his era, minimizing errors and ensuring the reliability of his data. In 1755 they published "De litteraria expeditione per pontificiam ditionem" (On the Scientific Expedition through the Papal States) that contained results of their survey and its analysis. For more details about the work of Boscovich, see Altić (2013). Stigler (1981) gives an exhaustive introduction to the history of regression.

The data on meridian arcs used by Boscovich was crucial in determining the shape and size of the Earth. He combined data from five locations:

Location	Latitude	ArcLength	sin2Latitude
Quito	0	56751	0
Cape of Good Hope	33	57037	2987
Rome	43	56979	4648
Paris	49	57074	5762
Lapland	66	57422	8386



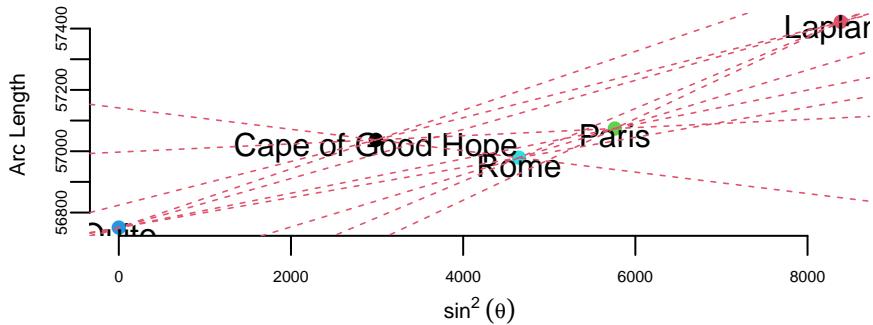
The arc length is measured in *toises*, a pre-metric unit of about 6.39 feet. Both the table and the plot show that arc length increases with latitude and that its relationship to $\sin^2 \theta$ is approximately linear:

$$\text{Arc Length} = \beta_0 + \beta_1 \sin^2 \theta$$

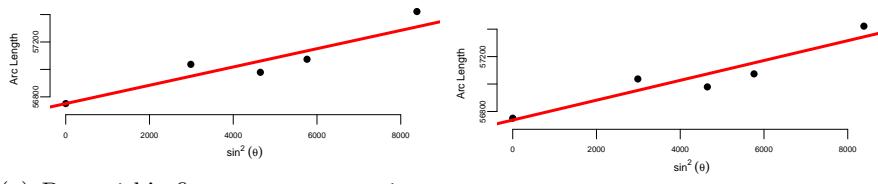
where θ is the latitude. Here β_0 is the length of a degree of arc at the equator, and β_1 is how much longer a degree of arc is at the pole. Boscovich faced a fundamental aggregation problem: how to distill five distinct, noisy measurements into single, optimal estimates for β_0 and β_1 . His first attempt to answer this question involved calculating ten slopes for each pair of points and then averaging them. The table below shows the ten slopes.

Table 10.2: Ten slopes for each pair of the five cities from the Boscovich data

	Quito	Cape of Good Hope	Rome	Paris	Lapland
Quito					
Cape of Good Hope	0.096				
Rome	0.049		-0.035		
Paris	0.056		0.013	0.085	
Lapland	0.080		0.071	0.118	0.13



The average of the ten slopes is 0.0667. Notice the slope between Cape of Good Hope and Rome is negative. This is due to the measurement error. Boscovich then calculated an average after removing this outlier. The average of the remaining nine slopes is 0.078. In both cases he used length of the arc at Quito as estimate of the intercept β_0 . Figure 10.1a shows the line that corresponds to the parameter estimates obtained by Boscovich. Figure 10.1b is the same plot but with the modern least squares line.



(a) Boscovich's first attempt to estimate the parameters (b) Modern least squares approach

Figure 10.1: Comparison of Boscovich's first attempt to estimate the parameters and the modern least squares approach

This is a very reasonable approach! However, Boscovich was not satisfied with it and sought a method that would minimize the sum of the absolute deviations between the data points and the fitted curve. Two years later, he developed a pioneering technique called *least absolute deviations*, which revolutionized data analysis. Unlike the later least squares approach, this method minimized absolute differences, making it particularly effective in handling measurement errors and outliers.

Armed with his meticulous measurements and innovative statistical analysis, Boscovich not only confirmed the oblate spheroid shape of the Earth but also refined its dimensions. His calculations yielded a more accurate value for the Earth's equatorial radius and the flattening at the poles, providing crucial support for Newton's theory of gravitation, which predicted this very shape.

Motivated by the analysis of planetary orbits and determining the shape of the Earth, later in 1805, Adrien-Marie Legendre (1752-1833) published the first clear and concise explanation of the least squares method in his book “*Nouvelles methodes pour la determination des orbites des cometes*”. The method of least squares is a powerful statistical technique used today to fit a mathematical model to a set of data points. Its goal is to find the best-fitting curve that minimizes the sum of the squared distances (also known as residuals) between the curve and the actual data points. Compared to the approach proposed by Boscovich, the least squares method is less robust to measurement errors and inconsistencies. However, from a computational point of view, it is more efficient and various algorithms exist for efficient calculation of curve parameters. This computational efficiency is crucial for modern data analysis, where datasets can be massive and complex, making least squares a fundamental tool in statistics and data analysis, offering a powerful and widely applicable approach to data fitting and model building.

Legendre provided a rigorous mathematical foundation for the least squares method, demonstrating its theoretical underpinnings and proving its optimality under certain conditions. This mathematical basis helped establish the credibility and legitimacy of the method, paving the way for its wider acceptance and application. Legendre actively communicated his ideas and collaborated with other mathematicians, such as Carl Friedrich Gauss (1777-1855), who also contributed significantly to the development of the least squares method. While evidence suggests Gauss used the least squares method as early as 1795, his formal publication came later than Legendre's in 1809. Despite the delay in publication, Gauss independently discovered the method and applied it to various problems, including celestial mechanics and geodesy. He developed efficient computational methods for implementing the least squares method, making it accessible for practical use by scientists and engineers. While Legendre's clear exposition and early publication brought the least squares method to the forefront, Gauss's independent discovery, theoretical development, practical applications, and contributions to computational methods were equally crucial in establishing the method's significance and impact. Both mathematicians played vital roles in shaping the least squares method into the powerful statistical tool it is today.

Another French polymath Pierre-Simon Laplace (1749-1827) extended the methods of Boscovich and showed that the curve fitting problem could be solved by ordering the candidate slopes and finding the weighted median. Besides that, Laplace made fundamental contributions to probability theory, developing the Bayesian approach to inference. Most of Laplace's work was in the field of celestial mechanics, where he used data from astronomical observations to develop mathematical models and equations describing the gravitational interactions between celestial bodies. His analytical methods and use of observational data were pioneering in the field of celestial mechanics. Furthermore, he developed methods for estimating population parameters from samples,

such as the mean and variance and pioneered the use of random sampling techniques, which are essential for ensuring the validity and generalizability of statistical inferences. These contributions helped lay the foundation for modern sampling theory and survey design, which are crucial for conducting reliable and representative studies. Overall, Laplace's contributions to data analysis were profound and enduring. His work in probability theory, error analysis, sampling methods, and applications significantly advanced the field and laid the groundwork for modern statistical techniques. He also played a crucial role in promoting statistical education and communication, ensuring that these valuable tools were accessible and utilized across various disciplines.

Boscovich used what we call today a linear regression analysis. This type of analysis relies on the assumption that the relationship between the independent (sine squared of the latitude) and dependent (arc length) variables is linear. Francis Galton was the person who coined the term “regression” in the context of statistics. One of the phenomena he studied was the relationship between the heights of parents and their children. He found that the heights of children tended to regress towards the average height of the population, which led him to use the term “regression” to describe this phenomenon. Galton promoted a data-driven approach to research that continues to shape statistical practice today. Furthermore, he introduced the concept of quantiles, which divide a population into equal-sized subpopulations based on a specific variable. This allowed for a more nuanced analysis of data compared to simply considering the mean and median. He also popularized the use of percentiles, which are specific quantiles used to express the proportion of a population below a certain value.

Galton used regression analysis to show that the offspring of exceptionally large or small sizes of sweet peas tended to be closer to the average size. He also used it for family studies and investigated the inheritance of traits such as intelligence and talent. He used regression analysis to assess the degree to which these traits are passed down from parents to offspring.

Galton’s overall approach to statistics was highly influential. He emphasized the importance of quantitative analysis, data-driven decision-making, and empirical research, which paved the way for modern statistical methods and helped to establish statistics as a recognized scientific discipline.

While Galton used data to understand heredity and regression to the mean in biological systems, modern applications leverage the same principles at a scale and speed he could scarcely have imagined. The evolution of statistical methods—from Boscovich’s manual calculations to Galton’s regression—has culminated in systems that perform these optimizations millions of times per second. In high-stakes environments, the “unreasonable effectiveness of data” transforms from a tool for scientific discovery into a competitive necessity.

Example 10.2 (Formula One: Data at 200 mph). Formula One represents the pinnacle of this transformation, where championships are increasingly de-

cided by algorithms rather than just driver reflex. As detailed in [Artificial Intelligence in Formula 1](#), modern F1 cars are essentially rolling data centers. With 300 sensors per car generating millions of data points over a 200-mile race, teams must optimize critical variables—fuel load, tire degradation, and weight effects—in real-time.

The key innovation is moving from pre-race strategy planning to in-race dynamic optimization using cloud computing platforms like AWS. Teams run Monte Carlo simulations of all cars and traffic situations to continuously update their strategy recommendations. Instead of following static pre-race plans, teams can now dynamically optimize pit stops, tire selection, and fuel loads in response to unfolding race conditions.

The article emphasizes that the best strategies can vary dramatically from moment to moment during a race, making real-time AI-powered decision making crucial for competitive advantage. Teams are limited to 60 data scientists, so they must rely heavily on automated machine learning systems to process the vast amounts of sensor data and generate actionable insights during races.

The [CNBC article](#) highlights how Formula One championships are increasingly being determined by technological innovation rather than just driver skill. F1 success depends heavily on advanced data analytics, machine learning algorithms, and real-time computing capabilities. Key technological factors driving F1 success include real-time data processing where teams process millions of data points from hundreds of sensors per car during races. AI-powered strategy optimization uses machine learning algorithms to continuously analyze race conditions and recommend optimal pit stop timing, tire choices, and fuel management. Cloud computing infrastructure allows teams to rely on platforms like AWS to run complex simulations and data analysis during races. Predictive modeling employs advanced algorithms to predict tire degradation, fuel consumption, and competitor behavior. Simulation capabilities enable teams to run thousands of Monte Carlo simulations to optimize race strategies.

The technological arms race in Formula One has led to significant regulatory challenges. To maintain competitive balance and prevent larger teams from gaining insurmountable advantages through unlimited technological investment, F1 has implemented strict caps on the number of engineers and data scientists that teams can employ. Currently, teams are limited to 60 data scientists and engineers, which forces them to be highly strategic about their technological investments and resource allocation. This cap creates an interesting dynamic where teams must balance the need for sophisticated AI and machine learning capabilities with the constraint of limited human resources. As a result, teams are increasingly turning to automated systems and cloud-based solutions to maximize their technological capabilities within these constraints. The cap also levels the playing field somewhat, ensuring that success depends more on the efficiency and innovation of the technology rather

than simply having more engineers and data scientists than competitors.

The Formula One example illustrates the power of data *quantity* and *velocity*—millions of data points processed instantly to optimize a single objective. However, the effectiveness of data depends just as heavily on how it is *aggregated*. Having massive amounts of data is useless if the method of combination introduces bias or fails to extract the signal. This brings us to a fundamental paradox in data science: under similar conditions, aggregating independent judgments can lead to either supernatural accuracy or catastrophic delusion.

10.1 The Wisdom and Madness of Crowds

“Men, it has been well said, think in herds; it will be seen that they go mad in herds, while they only recover their senses slowly, and one by one.” This observation opens Charles Mackay’s 1841 work *Extraordinary Popular Delusions and the Madness of Crowds*, one of the earliest systematic examinations of collective human behavior. Yet just sixty-six years later, Francis Galton would document the opposite phenomenon: a crowd of nearly 800 people accurately estimating the weight of an ox to within 1% of its true value. How can crowds be both mad and wise? And what does this duality tell us about aggregating data and building intelligent systems?

The tension between collective wisdom and collective madness cuts to the heart of modern data science and artificial intelligence. When we combine multiple predictions, aggregate diverse data sources, or build ensemble models, we implicitly trust in the wisdom of aggregation. Yet history—from the Dutch tulip mania of 1637 to the dot-com bubble of the late 1990s—reminds us that crowds can be spectacularly, catastrophically wrong. Understanding when and why crowds exhibit wisdom versus madness has profound implications for how we design AI systems, interpret market prices, and aggregate human judgment.

10.1.1 Historical Delusions and Economic Bubbles

Charles Mackay’s *Extraordinary Popular Delusions and the Madness of Crowds* (Mackay 1841) chronicled three major categories of collective folly: economic manias, alchemical and prophetic delusions, and the social dynamics of witch hunts and fortune-telling. His most enduring contribution lies in documenting economic bubbles, particularly three spectacular episodes from the 17th and 18th centuries.

The Dutch tulip mania of 1636-1637 represents perhaps the purest example of speculative madness. At the height of the mania, single tulip bulbs sold

for more than ten times the annual income of a skilled craftsman. A *Semper Augustus* bulb reportedly sold for 6,000 florins—enough to purchase one of the grandest houses on the most fashionable canal in Amsterdam. The market wasn’t driven by the intrinsic value of tulips or even their aesthetic beauty, but by the expectation that prices would continue rising. When the bubble inevitably collapsed in February 1637, fortunes vanished overnight, leaving behind ruined merchants and a cautionary tale about the dangers of collective speculation.

The South Sea Bubble of 1720 in England followed a similar trajectory but on a larger scale. The South Sea Company, granted a monopoly on trade with South America, saw its stock price rise from £128 in January to over £1,000 by August—despite generating minimal actual revenue from trade. The company’s success spawned a wave of similarly dubious ventures, including companies “for carrying on an undertaking of great advantage, but nobody to know what it is.” When confidence finally broke, the collapse devastated the British economy and ruined thousands of investors, including Isaac Newton, who reportedly lamented, “I can calculate the motion of heavenly bodies, but not the madness of people.”

The Mississippi Scheme in France, orchestrated by Scottish financier John Law, created the third great bubble Mackay chronicled. Law convinced the French government to establish a national bank and a trading company with monopoly rights to develop French Louisiana. Through a complex scheme of debt conversion and money printing, Law inflated both the currency and company shares to unsustainable levels. When the bubble burst in 1720, it destroyed the French economy and created such trauma that France wouldn’t establish another national bank for decades.

Mackay’s analysis identified common patterns across these episodes:

- *Gradual inception:* Bubbles begin with a kernel of truth—tulips were genuinely valuable, the South Sea Company did have a trade monopoly, Louisiana held economic potential.
- *Social contagion:* As early investors profit, others join not from fundamental analysis but from observing their neighbors’ gains.
- *Suspension of skepticism:* Normal risk assessment breaks down as stories of easy wealth dominate rational calculation.
- *New era thinking:* Participants convince themselves “this time is different,” that old rules no longer apply.
- *Catastrophic collapse:* Once confidence breaks, the crowd rushes for the exits, and prices collapse faster than they rose.

While modern historians have questioned some of Mackay’s details—the tulip mania may have been less extreme and more localized than he portrayed—his central insight endures: crowds can synchronize on beliefs wildly divorced from reality, sustaining collective delusions until the inevitable reckoning.

10.1.2 Galton's Ox: The Wisdom of Aggregation

Francis Galton, whom we met earlier pioneering regression and correlation, approached crowd behavior from a different angle. In 1907, he attended a livestock fair in Plymouth where nearly 800 people paid sixpence each to guess the dressed weight of an ox. The crowd included expert butchers and farmers as well as complete novices. After the competition, Galton obtained all the tickets and analyzed the distribution of guesses (Galton 1907).

The ox weighed 1,198 pounds when dressed. The median estimate from the crowd: 1,207 pounds—an error of less than 1%, or just 9 pounds. This remarkable accuracy led Galton to conclude that the result was “more creditable to the trustworthiness of a democratic judgment than might have been expected.”

Galton’s statistical analysis revealed several fascinating patterns. The probable error of a single randomly selected estimate was ± 37 pounds (about 3.1% of the true weight). Yet the median captured the true weight with far greater precision. The distribution wasn’t symmetric: estimates above the median had a quartile deviation of 45 pounds (3.7%), while estimates below deviated by only 29 pounds (2.4%). This asymmetry suggested systematic cognitive bias—people were more likely to overestimate than underestimate weight.

The middlemost 50% of estimates ranged from 1,178 to 1,252 pounds, a spread of 74 pounds around the true value. Galton observed that competitors appeared to “have erred normally in the first instance, and then to have magnified all errors that were negative and to have minified all those that were positive”—a remarkably prescient description of what modern psychologists would call anchoring and adjustment biases.

What made the crowd wise? Several conditions aligned:

Independence: Each person wrote their estimate privately, without conferring. There was no oratory, no group discussion, no opportunity for social influence to create correlated errors.

Diversity: The crowd mixed genuine experts (butchers who judged livestock daily) with laypeople relying on crude heuristics. This heterogeneity ensured errors weren’t systematically biased in the same direction.

Incentive alignment: The sixpence fee deterred frivolous guesses (as Galton noted, it was “sufficiently high to prevent much ‘practical joking’”), while prizes motivated genuine effort.

Appropriate aggregation: Galton chose the median rather than the mean, making the result robust to outliers and extreme estimates.

The experiment demonstrated a fundamental principle: *aggregating diverse, independent estimates can produce accuracy exceeding individual expertise*. No single person—not even the most experienced butcher—was as accurate as

the median of the crowd. The collective judgment extracted signal from the noise of individual errors.

10.1.3 Smart Money, Dumb Money: Learning from Crowds

If Galton showed how aggregation creates wisdom, Heaton and Polson's 2012 paper reveals how it can perpetuate madness (Heaton and Polson 2012). Their work examines financial markets where two types of traders coexist: "smart money" who know true probabilities, and "dumb money" who hold systematically incorrect beliefs. Like in Galton's ox-guessing crowd, market participants report their estimates by actively betting against each other, and prices reflect the *proportion* of money on each side—not a probability-weighted average of beliefs.

Suppose an asset can default or not default. Smart money knows the true default probability is 30%, while dumb money believes it's only 10%. If dumb money constitutes 45% of the market, they'll bet heavily against default, placing 45% of total market capital on "no default." Smart money, knowing default is more likely, bets the remaining 55% on default. The equilibrium price for "default" becomes 0.55—simply the proportion betting on that outcome—which overestimates the true 30% probability.

This creates systematic inefficiency. Dumb money consistently loses because they bet at unfavorable odds, while smart money earns predictable profits. Yet the market doesn't naturally correct this imbalance. Smart money can't arbitrage away all the mispricing because they have limited capital and betting against a bubble is fraught with risk; as the famous saying goes, "the market can remain irrational longer than you can remain solvent." The inefficiency persists. The central question is then *Can dumb money learn from market prices that they are the dumb money?*

This proves surprisingly difficult due to the *identification problem*. When a trader observes a market price, they see the aggregate result of all bets, but they cannot inherently distinguish their own contribution from the "smart" contribution.

Consider a market split 45-55. A trader in the 45% group knows there is disagreement. However, they face a symmetric ambiguity:

1. *Scenario A*: They are the "Smart Money" (minority insiders), and the 55% are the "Dumb Money" (overconfident herd).
2. *Scenario B*: They are the "Dumb Money" (contrarian fools), and the 55% are the "Smart Money" (rational majority).

The market price alone ($p = 0.55$) is often consistent with *both* scenarios, rendering the signal uninformative. A price of 0.55 could arise from Smart Money

knowing the truth, or Dumb Money pushing the price up. You cannot facilitate the distinction just by looking at the price. Heaton and Polson formalize this using Bayesian learning. To learn they are “dumb,” a trader must update their posterior odds:

$$\frac{P(\text{Dumb} \mid \text{Price})}{P(\text{Smart} \mid \text{Price})} = \underbrace{\frac{P(\text{Price} \mid \text{Dumb})}{P(\text{Price} \mid \text{Smart})}}_{\text{Likelihood Ratio}} \times \underbrace{\frac{P(\text{Dumb})}{P(\text{Smart})}}_{\text{Prior Odds}}$$

If the Likelihood Ratio is 1, no learning occurs. This happens when the observed price is equally probable whether you are smart or dumb.

In the worst-case scenario, the distributions of expected prices conditional on your type overlap perfectly.

Learning requires two conditions: (1) the likelihood ratio must favor being dumb money given the observed price, and (2) your prior must assign non-zero probability to being dumb. When smart and dumb money are equally balanced, the likelihood ratio equals 1. No amount of observation can update beliefs about one’s type. Paradoxically, increasing smart money toward this 50/50 balance might reduce market efficiency by making it harder for dumb money to learn.

As the authors note, “there is a sense in which the essence of being the dumb money is thinking too strongly that one is the smart money.” Learning requires assigning non-zero prior probability to being systematically wrong—a “psychologically difficult self-evaluation” that may be precisely what defines dumb money. Overconfidence isn’t just a symptom of being dumb money; it’s what prevents learning that one is dumb money.

Even if dumb money suspects they might be wrong, they must accurately estimate the proportion of dumb money in the market. But identifying market composition is notoriously difficult. During the dot-com bubble, for instance, dumb money “had been laughing all the way to the bank”—their short-term success reinforced confidence that they were actually the smart money.

Even if dumb money concludes they’re *probably* dumb money, learning must be strong enough to reverse their position. Merely adjusting beliefs slightly won’t change which side of the bet appears attractive. The paper illuminates why certain markets remain persistently inefficient. It’s not just that dumb money exists, but that *dumb money cannot learn it is dumb money*. The very characteristics that make someone dumb money—overconfidence, poor base rate estimation, unwillingness to question fundamental assumptions—are precisely those that prevent self-correction.

The authors cite the 2008 financial crisis as a real-world example. A handful of hedge funds bet against subprime mortgages at extremely favorable odds. As Michael Lewis documented in *The Big Short*, the constraint wasn’t demand

for the bet—smart money was willing—but supply. Finding enough counterparties willing to take the other side proved difficult. Yet even as the crisis unfolded, many “dumb money” participants couldn’t learn their type until catastrophic losses forced the realization.

The contrast between Galton’s wise crowd and Heaton and Polson’s mad market reveals critical lessons for building intelligent systems. Galton’s ox-guessing competition succeeded because it created ideal conditions for wisdom:

Statistical independence: Errors weren’t correlated. When one person overestimated, another’s underestimate balanced it out. This is why ensemble machine learning methods work: combining independent models reduces variance while preserving low bias. Random forests, for instance, decorrelate decision trees by training on random subsets of data and features, ensuring individual tree errors don’t reinforce each other.

Diversity of approach: The crowd used heterogeneous methods—some people estimated volume then converted to weight, others compared to familiar animals, butchers relied on professional experience. This diversity ensured systematic biases in one approach were offset by different biases in another. Similarly, ensemble methods benefit from combining fundamentally different model types (e.g., neural networks, tree-based models, and linear models) rather than multiple instances of the same architecture.

Appropriate aggregation: Galton used the median, which is robust to outliers. In modern ensemble methods, we similarly choose aggregation schemes carefully: majority voting for classification, median or trimmed mean for regression, weighted combinations based on confidence. The aggregation method matters as much as the individual models.

No strategic interaction: Participants weren’t betting against each other or trying to exploit others’ mistakes. Each estimate represented genuine belief. This differs fundamentally from adversarial settings where agents game the system.

Heaton and Polson’s market differs on crucial dimensions:

Systematic subgroup bias: Dumb money isn’t randomly wrong—they’re systematically wrong in the same direction. Aggregating their bets doesn’t cancel errors; it embeds bias in prices. In machine learning, if multiple models share the same systematic bias (say, all trained on biased data), ensembling won’t fix the problem. Voting among biased models just entrenches the bias.

Strategic interaction: Market participants bet against each other. Prices reflect not just beliefs but capital constraints and risk appetite. Smart money can’t arbitrage away all inefficiency. Similarly, in adversarial machine learning settings (spam detection, fraud detection, adversarial attacks on neural networks), the presence of strategic agents fundamentally changes aggregation dynamics.

Circular inference: Prices reflect participants' own bets, creating a circularity: dumb money observes prices that partially reflect their own behavior and must infer whether they're on the right or wrong side. In machine learning, this resembles the challenge of training on data that includes your model's own predictions—a form of feedback loop that can amplify rather than correct errors.

Barriers to self-correction: Dumb money cannot learn its type without assigning prior probability to being wrong and accurately estimating market composition. In machine learning, this parallels the challenge of model selection uncertainty: an algorithm must know which *class* of model is appropriate before it can learn parameters. Choosing the wrong model class can be more damaging than getting parameters slightly wrong.

Designing Robust AI Systems

These lessons suggest several principles for building intelligent systems. *Independence* is important. Training on different data subsets through bagging creates independence by ensuring each model sees a different sample of the data. Using different features, as in random forests, prevents models from making identical mistakes based on the same input patterns. Employing different algorithms through stacking combines fundamentally different approaches to the same problem. Adding noise through techniques like dropout and data augmentation decorrelates errors by introducing controlled randomness that prevents models from overfitting to identical patterns.

Calibrate confidence: Overconfidence is as dangerous in AI as in markets. Dumb money thinks it's smart money; overfit models are "confident but wrong." Calibration—ensuring predicted probabilities match actual frequencies—helps prevent this. Techniques like temperature scaling, Platt scaling, and isotonic regression (which enforces a monotonic relationship between predicted scores and probabilities) adjust model confidence to better reflect true uncertainty.

Avoid feedback loops: Be cautious when models influence the data they'll later train on. This occurs in recommender systems, where showing users content based on past behavior creates training data from that very behavior. Financial trading algorithms face similar challenges when their price predictions actually affect market prices. Search engines encounter this when their ranking algorithms influence user clicks, which then become training data for future ranking decisions. Content moderation systems create feedback loops when automated decisions generate the training data for future automation, potentially amplifying initial biases or errors.

Provide unambiguous feedback: Unlike markets where feedback is delayed and noisy, AI systems should enable rapid, clear feedback about prediction quality.

This accelerates learning and prevents prolonged periods of confident incorrectness.

The Bias-Variance Tradeoff

Galton's experiment beautifully illustrates the bias-variance tradeoff. Individual estimates had high variance (probable error of 3.1%) but low bias (median off by only 0.8%). The median reduced variance dramatically while preserving the low bias—a core principle in statistical learning.

Ensemble methods exploit the same principle. If individual models have low bias but high variance, averaging reduces variance without increasing bias. This explains why bagging (bootstrap aggregating) works so well: by training multiple models on random data subsets, we create high-variance predictors whose errors largely cancel when averaged.

However, if models have systematic bias, averaging won't help—it may even hurt. If all models in an ensemble underestimate values for a particular subgroup (perhaps due to underrepresentation in training data), taking their average still underestimates. This is the “dumb money” problem: when errors are correlated and biased in the same direction, aggregation entrenches rather than eliminates the problem.

Information Markets and Prediction Platforms

Modern prediction markets attempt to harness Galton's wisdom while avoiding Heaton and Polson's madness. Platforms like Metaculus, Good Judgment Open, and Manifold Markets aggregate forecasts from diverse participants to predict future events—from election outcomes to technological breakthroughs.

These platforms implement several design principles to promote wisdom over madness:

Proper scoring rules: Participants are rewarded for accuracy, not just correct predictions. The Brier score, for instance, penalizes both overconfidence and underconfidence, incentivizing honest reporting of beliefs rather than strategic betting.

Reputation systems: Track forecaster accuracy over time, weighting predictions by historical performance. This effectively filters out “dumb money” by giving less influence to consistently poor predictors.

Extremization: Research by Tetlock and others shows that aggregated predictions often benefit from “extremizing”—adjusting the consensus forecast away from 50% toward the extremes. If the average forecast is 70%, adjusting to 75% often improves accuracy. This suggests crowds are sometimes too cautious, insufficiently updating on shared information.

Transparency: Display the distribution of forecasts, not just the median. This reveals when consensus is strong versus when the crowd is divided, providing information about uncertainty in the aggregate.

Incentive alignment: Some platforms use real money (prediction markets), while others use reputation points or tournament prizes. The key is creating genuine incentive to be accurate rather than to follow the crowd or game the system.

Early evidence suggests these platforms can achieve impressive accuracy. Before the 2020 U.S. presidential election, prediction markets aggregated thousands of bets to estimate Biden's probability of victory at around 60-70%, roughly matching sophisticated polling models. During the COVID-19 pandemic, forecasting platforms predicted vaccine development timelines more accurately than many expert committees. The success of these platforms validates Galton's core insight: properly aggregated diverse judgments can rival or exceed expert predictions.

The arc from Mackay through Galton to Heaton and Polson traces the evolution of our understanding of collective intelligence. Mackay warned that crowds "go mad in herds," documenting the catastrophic consequences of synchronized delusion. Galton demonstrated that crowds can extract wisdom from noise through proper aggregation of independent judgments. Heaton and Polson revealed the subtle conditions under which madness persists—when systematic bias, strategic interaction, and barriers to learning prevent self-correction.

For AI and machine learning, these lessons are foundational. Every ensemble method, every data aggregation scheme, every model averaging technique implicitly bets on the wisdom of aggregation. But wisdom isn't automatic—it emerges from independence, diversity, appropriate aggregation methods, and absence of systematic bias. When these conditions fail, we get not intelligence but amplified error: overfit models that are confident but wrong, biased algorithms that entrench inequality, feedback loops that amplify rather than correct mistakes.

The unreasonable effectiveness of data depends not just on having more data, but on aggregating it wisely. As we build increasingly sophisticated AI systems that combine multiple models, integrate diverse data sources, and make consequential decisions, the distinction between wisdom and madness—between Galton's ox and Mackay's tulips—becomes ever more critical. The crowds we build into our algorithms must be wise ones, designed with intention to harness collective intelligence while guarding against collective delusion.

11

Pattern Matching

“*Prediction is very difficult, especially about the future.*” Niels Bohr, Danish physicist and Nobel laureate

The history of data analysis is closely intertwined with the development of pattern matching techniques. The ability to identify and understand patterns in data has been crucial for scientific discoveries, technological advancements, and decision-making. From the early days of astronomy to modern machine learning, pattern matching has played a pivotal role in advancing our understanding of the world around us. This chapter explores the key concepts of pattern matching, its historical development, and its impact on data analysis.

Data science involves two major steps: collection and cleaning of data and building a model or applying an algorithm. In this chapter we present the process of building predictive models. To illustrate the process, think of your data as being generated by a black box in which a set of input variables x go through the box and generate an output variable y .

11.1 Why Pattern Matching?

For Gauss, Laplace, and many other scientists, the central challenge was estimating parameters when the functional form of the relationship was already known—often linear (as in the Earth-shape example) or multiplicative (e.g., Newton’s $F = ma$ or Einstein’s $E = mc^2$). In many modern problems, however, the relationship itself is unknown and defies simple mathematical description; human behaviour and natural language are prominent examples (Halevy, Norvig, and Pereira (2009)).

In such cases a *pattern-matching* approach can uncover the hidden relationships directly from data. Pattern matching means identifying recurring sequences, relationships, or structures within a dataset—much like finding a puzzle piece that completes a picture. Recognising these patterns yields insights, reveals trends, supports prediction, and ultimately improves decisions.

Initial pattern-matching studies have often sparked major scientific advances, as the early history of mammography illustrates.

Example 11.1 (Mammography and Early Pattern Matching). Early mammography relied on visual pattern matching to detect cancer signs like masses and microcalcifications. Radiologists used their expertise to distinguish these patterns from normal tissue, though the process was subjective and error-prone. Despite these challenges, this visual pattern matching laid the foundation for modern screening.

German surgeon Albert Solomon pioneered this field with his 1913 monograph (Nicosia et al. (2023)). By comparing X-rays of surgically removed tissue with the actual specimens, he identified characteristic features of tumors and was among the first to link microcalcifications to breast cancer—a correlation that remains a key biomarker today, even as the underlying molecular mechanisms are still being studied (Bonfiglio et al. (2021)).

11.1.1 Richard Feynman on Pattern Matching and Chess

Richard Feynman, the renowned physicist, argued that many scientific discoveries begin with pattern matching—a skill experts develop to identify structures and regularities in their domain. He frequently engaged in discussions about artificial intelligence, often using chess as an analogy to illustrate the difference between human intuition and machine calculation.

Feynman observed that while novices calculate moves, masters recognize patterns. They understand the “laws” of the board much like physicists understand the laws of nature. In an interview, he described learning the rules of chess simply by observing games: first noting that bishops maintain their color, then realizing they move diagonally. This process of uncovering rules from observations is the essence of scientific discovery.

Regarding machine intelligence, Feynman was pragmatic. He noted that machines need not “think” like humans to achieve similar or superior results. Just as airplanes fly without flapping wings like birds, computers can play chess (or solve other problems) using different underlying mechanisms—such as massive calculation or statistical optimization—yet achieve superhuman performance.

“If we would like to make something that runs rapidly over the ground... we could try to make a machine that runs like a cheetah. But, it’s easier to make a machine with wheels... later machines are not going to think like people think.” — Richard Feynman

This distinction is crucial in modern AI. Today’s systems, like AlphaZero, combine pattern matching (via neural networks) with search (Monte Carlo simulation), effectively “learning” chess principles from scratch. They don’t

rely on human heuristics (like “control the center”) but discover their own statistical patterns that maximize the probability of winning.

Discussions with professional pianist Beatrice Rana revealed an interesting parallel. She compared modern AI’s ability to produce incredible results to a pianist capable of remembering and reproducing complex pieces of music—calling it “*intelligenza artigiana*”, or the intelligence of hands: a mastery of execution and pattern without necessarily possessing human-like consciousness.

How do we translate this conceptual “pattern matching” into concrete algorithms? In data science, we formalize this process by defining a mathematical structure (a model) and using data to adjust its parameters. Whether we are predicting election outcomes or classifying images, the core task remains the same: finding a function that maps inputs to outputs in a way that generalizes to new, unseen data.

We will now move from the intuitive understanding of pattern matching to its formalization in predictive modeling.

11.2 Supervised Learning

Prediction and forecasting are central challenges in data analysis, predominantly solved using pattern matching approaches. In business and engineering, the main motivation is to make better decisions; in science, to test and validate theories. By uncovering trends and patterns in historical data, analysts can make *informed decisions* about the future, identify *potential risks and opportunities*, and develop proactive strategies to capitalize on them. While unsupervised learning (clustering, dimensionality reduction) finds patterns without labeled data, this chapter focuses on *supervised learning*, where we have a target variable we wish to predict.

The problem of supervised learning is to learn patterns from observed data to make predictions on new, unseen data. The key idea is that we have input-output pairs (x_i, y_i) where we know the correct output y_i for each input x_i , and we use these examples to learn a function that maps inputs to outputs.

Supervised learning powers a surprising range of modern systems. Manufacturing plants use sensor data to predict equipment failures before they happen. Autonomous vehicles rely on it for object detection, lane recognition, and real-time decision-making from camera and LiDAR feeds. Medical imaging systems detect diseases from X-rays and MRIs with accuracy rivaling radiologists. Banks use it for fraud detection and credit scoring; retailers for demand

forecasting and inventory optimization; marketing teams for churn prediction and lead scoring.

The common thread: you have historical data with known outcomes, and you want to predict outcomes for new cases. The rest of this chapter develops the mathematical framework for doing this well.

A typical prediction problem involves building a rule that maps observed inputs x into the output y . The inputs x are often called predictors, features, or independent variables, while the output y is often called the response or dependent variable. The goal is to find a predictive rule

$$y = f(x).$$

The map f can be viewed as a black box which describes how to find the output y from the input x . One of the key requirements of f is that we should be able to efficiently find this function using an algorithm. In the simple case y and x are both univariate (scalars) and we can view the map as

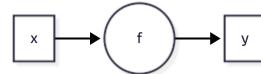


Figure 11.1: Black box model

The goal of machine learning is to reconstruct this map from observed data. In a multivariate setting $x = (x_1, \dots, x_p)$ is a list of p variables. This leads to a model of the form $y = f(x_1, \dots, x_p)$. There are a number of possible goals of analysis, such as estimation, inference or prediction. The main one being prediction.

The prediction task is to calculate a response that corresponds to a new feature input variable. An example of inference is the task of establishing causation, with the goal of extracting information about the nature of the black box association of the response variable to the input variables.

In either case, the goal is to use data to find a pattern that we can exploit. The pattern will be “statistical” in its nature. To uncover the pattern we use a training dataset, denoted by

$$D = (y_i, x_i)_{i=1}^n$$

where x_i is a set of p predictors and y_i is response variable. Prediction problem is to use a training dataset D to design a rule that can be used for predicting output values y for new observations x .

Let $f(x)$ be predictor of y , we will use notation

$$\hat{y} = f(x).$$

To summarize, we will use the following notation.

y	output variable (response/outcome)
x	input variable (predictor/covariate/feature)
$f(x)$	predictive rule
\hat{y}	predicted output value

We distinguish several types of input or output variables. First, *binary* variables that can only have two possible values, e.g. yes/no, left/right, 0/1, up/down, etc. A generalization of binary variable is a *categorical* variable that can take a fixed number of possible values, for example, marriage status. Additionally, some of the categorical variable can have a natural order to them, for example education level or salary range. Those variables are called *ordinal*. Lastly, the most common type of a variable is *quantitative* which is described by a real number.

Depending on the type of the output variable, there are three types of prediction problems.

Table 11.2: Types of output variables and corresponding prediction problems.

Output Variable Type	Description	Prediction Problem
Binary	$y \in \{0, 1\}$	Classification
Categorical	$y \in \{0, \dots, K\}$ for K possible categories	Classification
Quantitative	$y \in \mathbb{R}$ (any real number)	Regression
Ordinal	y has natural ordering	Ranking

Here are some examples of prediction problems:

Binary Classification: Predicting whether an email is spam or not spam involves input variables such as email content, sender information, presence of certain keywords, and email length. The output variable is $y \in \{0, 1\}$ where 0 = not spam, 1 = spam. The goal is to classify new emails as spam or legitimate.

Categorical Classification: Predicting the type of social media content based on text and image features uses input variables including text content, image features, user engagement metrics, posting time, and hashtags. The output variable is $y \in \{0, 1, 2, 3, 4\}$ where 0 = news, 1 = entertainment, 2 = educational, 3 = promotional, 4 = personal. The goal is to automatically categorize social media posts for content moderation and recommendation systems.

Regression (Quantitative): Predicting house prices based on features uses input variables such as square footage, number of bedrooms, location, age of house, and lot size. The output variable is $y \in \mathbb{R}$ (house price in dollars). The goal is to predict the selling price of a new house.

Ranking (Ordinal): Predicting customer satisfaction ratings involves input variables including product quality, customer service experience, delivery time, and price. The output variable is $y \in \{1, 2, 3, 4, 5\}$ where 1 = very dissatisfied, 5 = very satisfied. The goal is to predict customer satisfaction level for new customers.

There are several simple predictive rules we can use to predict the output variable y . For example, in the case of regression problem, the simplest rule is to predict the average value of the output variable. This rule is called the *mean rule* and is defined as

$$\hat{f}(x) = \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i.$$

Notice, this model does not depend on the input variable x and will predict the same value for all observations. This rule is simple and easy to implement, but it is not very accurate. In case of binary y , we can apply thresholding to the mean rule to obtain a binary classifier.

$$f(x) = \begin{cases} 1 & \text{if } \bar{y} > 0.5, \\ 0 & \text{if } \bar{y} \leq 0.5. \end{cases}$$

A more sophisticated rule is the *nearest neighbor rule*. This rule predicts the output value y for a new observation x by finding the closest observation in the training dataset and using its output value. The nearest neighbor rule is defined as

$$f(x) = y_{i^*},$$

where

$$i^* = \arg \min_{i=1,\dots,n} \|x_i - x\|$$

is the index of the closest observation in the training dataset. These two models represent two extreme cases of predictive rules: the mean rule is “stubborn” (it always predicts the same value) and the nearest neighbor rule is “flexible” (can be very sensitive to small changes in the inputs). Using the language of statistics the mean rule is of high bias and low variance, while the nearest neighbor rule is of low bias and high variance. Although those two rules are simple, they sometimes lead to useful models that can be used in practice. Further, those two models represent a trade-off between accuracy and complexity (the bias-variance trade-off). We will discuss this trade-off in more detail in the later section.

The mean model and nearest neighbor model belong to a class of so-called *non-parametric* models. The non-parametric models do not make explicit assumption about the form of the function $f(x)$. In contrast, parametric models assume that the predictive rule $f(x)$ is a specific function defined by vector of parameters, which we will denote as θ . A typical notation is then

$$f_{\theta}(x).$$

Traditional modeling culture employs statistical models characterized by single-layer transformations (transforming inputs directly into outputs without intermediate hidden layers), where the relationship between input variables and output is modeled through direct, interpretable mathematical formulations. These approaches typically involve linear combinations, additive structures, or simple nonlinear transformations that maintain analytical tractability and statistical interpretability. The list of widely used models includes:

Table 11.3: Traditional statistical models.

Model	Formula	Parameters / Hyperparameters
Linear Regression	$y = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$	$\theta = (\beta_0, \beta_1, \dots, \beta_p)$
Generalized Linear Model (GLM)	$y = f^{-1}(\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p)$	$\theta = (\beta_0, \beta_1, \dots, \beta_p)$
Generalized Additive Model (GAM)	$y = \beta_0 + f_1(x_1) + \dots + f_k(x_k)$	$\theta = (\beta_0, f_1, \dots, f_k)$
Principal Component Regression (PCR)	$y = \beta^T(Wx), \quad W \in \mathbb{R}^{k \times p}, \quad k < p$	$\theta = (\beta, W)$
k-Nearest Neighbors (KNN)	$y = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$	k (neighbors count)

We wish to find map f such that

$$\begin{aligned} y &= f(x) \\ y &= f(x_1, \dots, x_p) \end{aligned}$$

Essentially, the goal is to perform the pattern matching, also known as non-parametric regression. It involves finding complex relationships in data without assuming a specific functional form.

11.3 Complex Functions

In contrast to single-layer approaches, Deep Learning employs sophisticated high-dimensional multi-layer neural network architectures that can capture complex, non-linear relationships in data through hierarchical feature learning. In deep learning, we use composite functions rather than additive functions. We write the superposition of univariate functions as

$$f = f_1 \circ \dots \circ f_L \text{ versus } f_1 + \dots + f_L$$

where composition $f = f_L(f_{L-1}(\dots f_1(x)))$ creates a “deep” hierarchical structure, as opposed to the “flat” additive structure of models like GAMs.

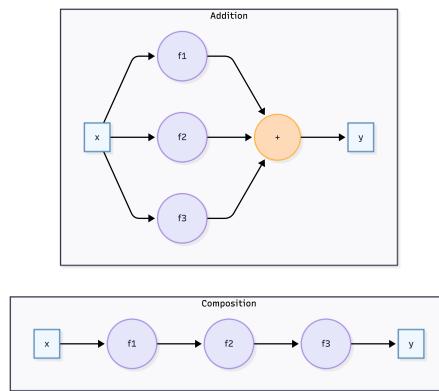


Figure 11.2: Composition vs Addition of Functions

Each function f_i in the composition is typically a combination of a linear transformation and a non-linear activation function

$$f_i(x) = \sigma(W_i x + b_i),$$

The set of parameters that we need to find is $\theta = (W_1, b_1, \dots, W_L, b_L)$. The depth and complexity of these architectures allow deep learning models to automatically discover intricate patterns—such as edges in images or grammar in text—from raw input data.

11.4 Model Estimation

There are two main approaches to finding the set of parameters θ . The first is optimization approach that minimizes a loss function. Loss function measures how well predictive rule f captures the relationship between input and output variables. The most common loss function is the mean squared error (MSE). The second approach is to use full Bayesian inference and to calculate the distribution over parameter θ given the observed data.

From the perspective of representation, feature engineering can be viewed as the search for low-dimensional summaries of x that retain the information needed for prediction: an informal echo of the role of sufficient statistics in classical inference (Chapter 3).

Both approaches start with formulating likelihood function. Likelihood is a function that tells us how probable the observed data is, given a particular value of the parameter in a statistical model. It is not the same as probability; instead, it's a function of the parameter, with the data fixed. Suppose you flip a biased coin 10 times and get 7 heads. You want to estimate the probability of getting heads on a single toss. You try different values of θ and ask: "How likely is it to get exactly 7 heads out of 10 flips if the true probability is θ ?" This leads to the likelihood function. Formally, given $y_i \sim f(y_i | x_i, \theta)$ as exchangeable (often simplified to i.i.d.) samples from a distribution with parameter θ , the likelihood function is defined as

$$L(\theta) = \prod_{i=1}^n p(y_i | x_i, \theta).$$

It treats the data $D = (y_i, x_i)_{i=1}^n$ as fixed and varies θ .

Likelihood connects our model to the data generating process by quantifying how likely it is to observe the actual data we have under different parameter values. For example, if we assume our data follows a normal distribution $y \sim N(f_\theta(x), \sigma^2)$ with mean $f_\theta(x)$ and variance σ^2 , the likelihood function would be:

$$L(\theta) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f_\theta(x_i))^2}{2\sigma^2}\right). \quad (11.1)$$

For the case of classification problem, we assume that y_i follows a Bernoulli distribution $y_i \sim \text{Bernoulli}(p_i)$. The likelihood function is defined as

$$L(\theta) = \prod_{i=1}^n p_i^{y_i} (1 - p_i)^{1-y_i}.$$

Here p_i is the probability of the response variable taking on a value of 1, given the input variables. A typical approach to calculate p_i is to use logistic function $\sigma(\cdot)$

$$f_\beta(x_i) = \beta^T x_i$$

$$p_i = \sigma(f_\beta(x_i)) = \frac{e^{f_\beta(x_i)}}{1 + e^{f_\beta(x_i)}},$$

Notice, that logistic function $\sigma(\cdot)$ is restricted to output values in $(0, 1)$.

The optimization-based approach is to find the set of parameters θ that maximizes the likelihood function.

$$\theta^* = \arg \max_{\theta} L(\theta).$$

Although most often, it is easier to optimize the log-likelihood function. We define the log-likelihood by

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^n \log p(y_i | x_i, \theta).$$

Notice that the log-likelihood is a sum of per-observation contributions, which is convenient for both analysis and computation. In many estimation problems we will instead minimize the negative log-likelihood (which plays the role of a loss),

$$l(\theta) = -\ell(\theta),$$

which is called the **cross-entropy** loss function.

Why does the solution not change? Since the logarithm is a monotonically increasing function, if $L(\theta_1) > L(\theta_2)$, then $\log L(\theta_1) > \log L(\theta_2)$. This means that the parameter value that maximizes the likelihood function will also maximize the log-likelihood function. The maximum point stays the same, just the function values are transformed.

The value of parameters θ that maximizes the log-likelihood is called the *maximum likelihood estimate* (MLE).

Now, rather than maximizing the log-likelihood function, we minimize the negative log-likelihood function

$$\theta^* = \arg \min_{\theta} l(\theta).$$

This problem is called the least squares problem.

Then the negative log-likelihood function is called the *loss function*. Thus the problem of finding maximum likelihood estimate is equivalent to minimizing the loss function.

Let's calculate the loss function that corresponds to the normal likelihood function given by Equation 11.1. Using the fact that the logarithm of a product is a sum of logarithms, we can write the negative log-likelihood as

$$l(\theta) = -\sum_{i=1}^n \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f_\theta(x_i))^2}{2\sigma^2}\right).$$

Inside the sum, we have a product of two terms. The first term is a constant with respect to θ and the second term is a function of θ . We can rewrite the likelihood function as

$$l(\theta) = -\sum_{i=1}^n \left[\log \frac{1}{\sqrt{2\pi\sigma^2}} + \log \exp\left(-\frac{(y_i - f_\theta(x_i))^2}{2\sigma^2}\right) \right].$$

The first term $\log \frac{1}{\sqrt{2\pi\sigma^2}}$ is a constant with respect to θ , so we can drop it from the optimization problem. The second term can be simplified using the fact that $\log \exp(x) = x$:

$$l(\theta) = \sum_{i=1}^n \left[\frac{(y_i - f_\theta(x_i))^2}{2\sigma^2} \right] + C,$$

where C is a constant that does not depend on θ . Since we are minimizing $l(\theta)$, we can drop constant terms that do not depend on θ :

$$l(\theta) = \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - f_\theta(x_i))^2.$$

This is the *mean squared error (MSE)* loss function, which is the most commonly used loss function for regression problems. The factor $\frac{1}{2\sigma^2}$ is often absorbed into the learning rate or regularization parameter in optimization algorithms. Thus, another name of the estimator is the *least squares estimator*. It is the same as the maximum likelihood estimator, assuming that the $f_\theta(x_i)$ is normally distributed.

11.4.1 Penalized Likelihood

While maximum likelihood estimation provides a principled approach to parameter estimation, we can often find better estimators using what is called a penalized likelihood. In fact, there are certain cases, when penalized estimator leads to universally better estimators. In statistics, we would say that MLE is inadmissible in dimensions of 3 or higher, meaning there exists another estimator (like the James-Stein estimator) that is strictly better in terms of expected

squared error (risk) by “shrinking” estimates towards a central value. Later in Chapter 17 we will discuss the theory of penalized estimators in more detail.

Penalized likelihood addresses overfitting by adding a regularization term to the likelihood function. Instead of maximizing just the likelihood, we maximize:

$$L_{\text{penalized}}(\theta) = L(\theta) \cdot \exp(\lambda\phi(\theta))$$

Or equivalently, we minimize the negative log-likelihood plus a penalty:

$$l(\theta) = \sum_{i=1}^n l(y_i, f_\theta(x_i)) + \lambda \sum_{j=1}^p \phi(\theta_j),$$

where $\lambda > 0$ is the regularization parameter that controls the strength of regularization, and $\phi(\theta)$ is the penalty function that measures model complexity. In machine learning the technique of adding the penalty term to the loss function is called regularization.

Regularization can be viewed as constraint on the model space. The techniques were originally applied to solve ill-posed problems where a slight change in the initial data could significantly alter the solution. Regularization techniques were then proposed for parameter reconstruction in a physical system modeled by a linear operator implied by a set of observations. It had long been believed that ill-conditioned problems offered little practical value, until Tikhonov published his seminal paper Andrey Nikolayevich Tikhonov et al. (1943) on regularization. Andrei N. Tikhonov (1963) proposed methods for solving regularized problems. In our notation, this corresponds to finding parameters θ that minimize

$$\min_{\theta} \|y - X\theta\|_2^2 + \lambda \|(\theta - \theta^{(0)})\|_q^q.$$

Here λ is the weight on the regularization penalty and the ℓ_q -norm is defined by $\|\theta\|_q = (\sum_i |\theta_i|^q)^{1/q}$. This optimization problem is a Lagrangian form of the constrained problem given by

$$\text{minimize}_{\theta} \quad \|y - X\theta\|_2^2 \quad \text{subject to } \sum_{i=1}^p \phi(\theta_i) \leq s.$$

with $\phi(\theta_i) = |\theta_i - \theta_i^{(0)}|^q$.

Later, sparsity became a primary driving force behind new regularization methods. The idea is that the vector of parameters θ is sparse, meaning that most of its elements are zero. This is a natural assumption for many models, such as the linear regression model. We will discuss the sparsity in more detail later in the book.

There are multiple optimization algorithms that can be used to find the solution to the penalized likelihood problem. Later in the book we will discuss the Stochastic Gradient Descent (SGD) algorithm, which is a popular tool for training deep learning models.

11.4.2 Bayesian Approach

Similar to the likelihood maximization approach, the Bayesian approach to model estimation starts with the likelihood function. The difference is that we assume that the parameters θ are random variables and follow some prior distribution. Then we use the Bayes rule to find the posterior distribution of the parameters, given the data $D = (y_i, x_i)_{i=1}^n$:

$$p(\theta | D, \lambda) \propto L(\theta)p(\theta | \lambda),$$

where $p(\theta | \lambda)$ is the prior distribution with hyperparameter λ and $L(\theta)$ is the likelihood function. It is a distribution over the parameters, not a single value.

Penalized likelihood has a natural Bayesian interpretation. The penalty term corresponds to a prior distribution on the parameters:

$$p(\theta) = \frac{1}{Z(\lambda)} \exp(-\lambda\phi(\theta))$$

Then the penalized likelihood is proportional to the posterior distribution:

$$p(\theta | y) \propto p(y|\theta) \cdot p(\theta) = L(\theta) \frac{1}{Z(\lambda)} \exp(-\lambda\phi(\theta))$$

This means maximizing the penalized likelihood is equivalent to finding the maximum a posteriori (MAP) estimate, which is the mode of the posterior distribution.

11.5 Prediction Accuracy

After we fit our model and find the optimal value of the parameter θ , denoted by $\hat{\theta}$, we need to evaluate the accuracy of the predictive model. Once $\hat{\theta}$ is obtained, it involves comparing the model's predictions to actual outcomes. We can simply use the value of the loss function from the training step to evaluate model's predictive power. However, this only tells us how well the model fits the training data. It doesn't tell us how well the model will perform

on unseen data. To evaluate the model's performance on unseen data, we need to use a different approach.

The most common approach is to split the data into training and test sets. The training set is used to train the model, while the test set is used to evaluate its performance. This approach is known as the train-test split. It is a simple and effective way to evaluate how well model predicts for unseen inputs.

Another approach is to use **cross-validation**. It involves splitting the data into smaller subsets and using them to train and test the model multiple times. When our sample size is small, this allows for a more robust estimate of the model's performance than simply splitting the data into a single training and test set. For small data sets, simple train-test split approach will be sensitive to choice of test samples, thus the estimated predicted performance will be unstable (high variance). Cross-validation helps to reduce this variance by averaging the performance across multiple folds. This makes the performance estimate more robust and less sensitive to the choice of test samples.

Cross-validation involves several steps. The data is randomly divided into k equal-sized chunks (folds). For each fold, the model is trained on $k - 1$ folds and tested on the remaining fold. This process is repeated k times, ensuring each fold is used for testing once. The performance of the model is evaluated on each fold using a chosen metric, such as accuracy, precision, recall, or F1 score. The average of the performance metrics across all k folds is reported as the final estimate of the model's performance.

A common choice for k is 5 or 10. When $k = n$ (where n is the sample size), this is known as leave-one-out cross-validation. This method can be computationally expensive but is less likely to overfit the data. Stratified cross-validation ensures that each fold contains approximately the same proportion of each class as in the entire dataset. This is important for imbalanced datasets where one class is significantly larger than the others.

Notice, that cross-validation requires re-training the model multiple times, which can be computationally expensive. Thus, for large datasets, we typically prefer simple train-test split. However, for small datasets, cross-validation can provide a more robust estimate of the model's performance.

Either method is limited to evaluating the model's performance on data that is available to the modeler. What if we start using our model on data that is different from the training and test sets? Unlike physics, where models often represent universal laws, data science deals with data generated by processes that may vary across contexts. For example, if we are building a model to predict the price of a house, we can train and test the model on data from a specific city. However, if we start using the model to predict the price of a house in a different city, the model might not perform as well. This is because the data from the new city might be different from the data used to train and test the model. This is known as the problem of generalization. It refers to the

ability of a model to perform well on data that is different from the training and test sets.

11.5.1 Evaluation Metrics for Regression

There are several metrics that can be used to evaluate the performance of regression models. We can simply use the same function as we use for fitting the model, e.g. least squares

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^n (y_i - \hat{y}_i)^2,$$

here \hat{y}_i is the predicted value of the i-th data point by the model $\hat{y}_i = f(x_i, \hat{\theta})$ and m is the total number of data points used for the evaluation. This metric is called the *Mean Squared Error (MSE)*. It is the average squared difference between the actual and predicted values. Lower MSE indicates better model performance, as it means the model's predictions are closer to the actual values.

A slight variation of this metric is Root Mean Squared Error (RMSE). This is the square root of MSE and is also commonly used due to its units being the same as the target variable.

$$\text{RMSE} = \sqrt{\text{MSE}}.$$

However, MSE is sensitive to outliers, as it squares the errors, giving more weight to large errors. This can lead to misleading results when the data contains outliers.

Mean Absolute Error (MAE) solves the sensitivity to the outliers problem. It is the mean of the absolute errors, providing a more robust measure than MSE for skewed error distributions

$$\text{MAE} = \frac{1}{m} \sum_{i=1}^n |y_i - \hat{y}_i|.$$

A variation of it is the *Mean Absolute Percentage Error (MAPE)*, which is the mean of the absolute percentage errors

$$\text{MAPE} = \frac{1}{m} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|.$$

Why R^2 Should Be Avoided

The *coefficient of determination* $R^2 = 1 - \sum(y_i - \hat{y}_i)^2 / \sum(y_i - \bar{y})^2$ is often misused as a measure of model quality. As Shalizi argues in [The Truth About Linear Regression](#), R^2 fundamentally does not measure goodness of fit:

- **R^2 can be arbitrarily low when the model is completely correct.** By having high noise variance σ^2 or low predictor variance $\text{Var}[X]$, we can drive R^2 toward 0 even when every assumption of the linear model holds perfectly.
- **R^2 can be arbitrarily close to 1 when the model is totally wrong.** A nonlinear, heteroskedastic model can produce $R^2 \approx 0.8$ simply because a tilted line fits better than a flat one—not because the linear model is correct.

Furthermore, R^2 says nothing about prediction error, cannot be compared across datasets, and the “variance explained” interpretation is misleading since regressing Y on X gives the same R^2 as regressing X on Y . Use MSE, MAE, or cross-validation error instead.

Finally, we can use graphics to evaluate the model’s performance. For example, we can create a scatterplot of the actual and predicted values of the target variable to visually compare them. We can also plot the histogram or a boxplot of the residuals (errors) to see if they are normally distributed.

11.5.2 Evaluation Metrics for Classification

Accuracy is the most fundamental metric used to evaluate models. It is defined as the ratio of the number of correct predictions to the total number of predictions. The formula is given by

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}},$$

where TP, TN, FP, and FN are the numbers of true positives, true negatives, false positives, and false negatives, respectively. However, it can be misleading for imbalanced datasets where one class is significantly larger than others. For example, if 95% of the data belongs to one class, a model that always predicts this class will be 95% accurate, even though it’s not very useful.

A more comprehensive understanding of model performance can be achieved by calculating the sensitivity (precision) and specificity (recall) as well as confusion matrix discussed in Section 2.5. The confusion matrix is

Actual/Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Precision measures the proportion of positive predictions that are actually positive. It is useful for evaluating how good the model is at identifying true positives. *Recall* measures the proportion of actual positives that are correctly

identified by the model. It is useful for evaluating how good the model is at not missing true positives.

Then we can use those to calculate *F1 Score* which is a harmonic mean of precision and recall, providing a balanced view of both metrics. The formula is given by

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Higher F1 score indicates better overall performance. If misclassifying certain instances is more costly than others, weighted metrics account for these different costs. For imbalanced datasets, metrics like F1 score or balanced accuracy are important to avoid misleading interpretations.

Sometimes, we use multiple metrics to get a comprehensive assessment of the model's performance. Additionally, consider comparing the model's performance to a baseline model or other existing models for the same task. Sometimes, it is hard to beat a "coin flip" classification model, when the model predicts the class randomly with equal probability. In regression, a simple baseline model is $f(x_i) = \bar{y}$, which is the mean of the target variable.



12

Linear Regression

The simplest form of a parametric model is a linear model that assumes a linear relationship between the input variables and the output variable. There are several ways to specify such a family of functions, for example as linear combinations of inputs

$$f_{\beta}(x) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p.$$

Technically, this is not a linear function. In mathematics, it is called an affine function, which is a linear function with an additional constant term (β_0). However, if we add a dummy variable $x_0 = 1$ to the input vector x , we can rewrite the function as a linear function of the parameters $\beta = (\beta_0, \beta_1, \dots, \beta_p)$ using vector notation

$$f_{\beta}(x) = \beta^T x, \quad x = (1, x_1, \dots, x_p).$$

Note that instead of using a generic notation θ for the parameters, we use β to emphasize that we are talking about the parameters of the linear model. The vector β is called the *regression coefficients*.

Regression analysis is the most widely used statistical tool for understanding relationships among variables. It provides a conceptually simple method for investigating functional relationships between one or more factors and an outcome of interest. This relationship is expressed in the form of an equation, which we call the model, connecting the response or dependent variable and one or more explanatory or predictor variables.

A more general form of a linear model is a linear combination of basis functions

$$f_{\beta}(x) = \beta_0 + \beta_1 \psi_1(x) + \dots + \beta_M \psi_M(x) = \beta^T \psi(x),$$

where ψ_1, \dots, ψ_M are the basis functions and $\psi(x) = (1, \psi_1(x), \dots, \psi_M(x))$.

Notice in the latter case, the function f is *linear in the parameters* $\beta = (\beta_0, \beta_1, \dots, \beta_M)$ but not in the input variables x . The goal of the modeler is to choose an appropriate set of predictors and basis functions that will lead to a good reconstruction of the input-output relations. After we've specified what the function f is, we need to find the best possible values of parameters β .

Finding a predictive rule $f_\beta(x)$ starts by defining the criterion of what is a good prediction. We assume that the function $f_\beta(x)$ is parameterized by a vector of parameters β and we want to find the best value of β that will give us the best prediction.

We will use a loss function that quantifies the difference between the predicted and actual values of the output variable. It is closely related to the loss function used in decision theory (thus the name). In decision theory, a loss function is a mathematical representation of the “cost” associated with making a particular decision in a given state of the world. It quantifies the negative consequences of choosing a specific action and helps guide decision-makers towards optimal choices. You can think of the loss function in predictive problems as a “cost” associated with making an inaccurate prediction given the values of the input variables x .

The *least squares* loss function, discussed in Chapter 11, is the sum of squared differences between the predicted and actual values. Given observed data set $\{(x_1, y_1), \dots, (x_n, y_n)\}$, the least squares estimator is the value of β that minimizes the sum of squared errors. In most regression settings we treat the observations as exchangeable (often simplified to i.i.d.), meaning the order of the data is not informative (Chapter 3).

$$\min_{\beta} \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad \hat{y}_i = \beta^T \psi(x_i).$$

In the unconditional case, when we do not observe any inputs x , the least squares estimator is the sample mean. We can solve this minimization problem by taking the derivative of the loss function and setting it to zero

$$\frac{d}{d\beta} \sum_{i=1}^n (y_i - \beta_0)^2 = -2 \sum_{i=1}^n (y_i - \beta_0) = 0$$

which gives us the solution $\hat{\beta}_0 = 1/n \sum_{i=1}^n y_i$, which is the sample mean.

Example 12.1 (Housing Price Prediction). To demonstrate linear regression we develop a model that predicts the price of a house, given its square footage. This is a simplified version of a regression model used by Zillow, a house pricing site. First, we look at property sales data where we know the price and some observed characteristics. Let’s look at the scatter plot of living areas measured in square feet and the price measured in thousands of dollars.

Second, we build a model that predicts price as a function of the observed characteristics. Now, we need to decide on what characteristics do we use. There are many factors or variables that affect the price of a house, for example location. Some other basic ones include size, number of rooms, and parking.

We will use the `SaratogaHouses` dataset from the `mlbench` package. This dataset contains information about house sales in Saratoga County, New

York, between 2004 and 2007. The dataset contains 1,172 observations and 16 variables. We will use the `price` variable as the dependent variable and the `livingArea` variable as the independent variable. The `price` variable contains the sale price of the house in thousands of dollars, and the `livingArea` variable contains the living area of the house in square feet.

$$\text{price} = \beta_0 + \beta_1 \text{livingArea}$$

The value that we seek to predict, `price`, is called the dependent variable, and we denote this by y . The variable that we use to construct our prediction, `livingArea`, is the predictor variable, and we denote it with x .

First, we examine the distribution of the data.

We use `lm` function to estimate the parameters of the line

This R code is fitting a linear regression model using the `lm` function. Let's break down the code:

1. `lm`: This is the function used for fitting linear models, it uses least squares loss function.
2. `price`: This is the dependent variable, the variable you are trying to predict. It is assumed to be in the dataset specified by the `data` argument.
3. `~`: In the context of the formula inside `lm`, the tilde (`~`) separates the dependent variable from the independent variable(s).
4. `livingArea`: This is the independent variable or predictor. In this case, the model is trying to predict the variable `price` based on the values of `livingArea`.
5. `data = d`: This specifies the dataset that contains the variables `price` and `livingArea`. The dataset is named `d`.

So, in plain English, this R code is fitting a linear regression model where the `price` is predicted based on the `livingArea` in the dataset `d`. The model is trying to find a coefficient for `livingArea` to minimize the difference between the predicted values and the actual values of `price`.

We can use `coef` function to find out the slope and the intercept of this line.

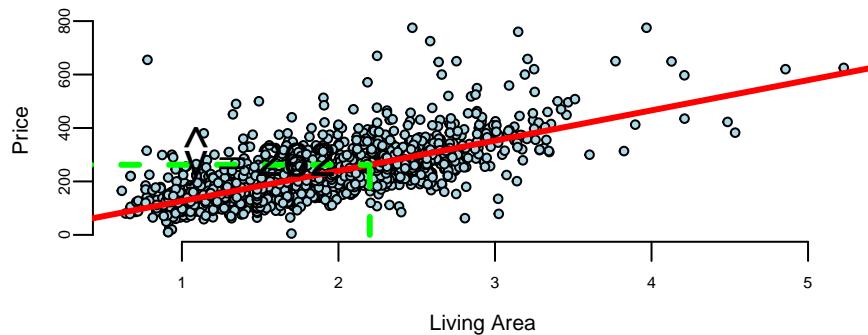
```
## (Intercept) livingArea
##           13          113
```

The intercept value is in units of y (\$1000). The slope is in units of y per units of x (\$1000/1 sq. feet). In our housing example, we find that $\beta_0 = 13.439394$ and $\beta_1 = 113.1225418$. Thus, every 1000 square feet increase the price of the house by \$113122.5418468 and the price of a lot of land without a house is

13.439394. The magnitude of β_1 shows the *economic significance* of house's square footage.

We can now predict the price of a house when we only know that size: take the value off the regression line. For example, given a house size of $x = 2.2$. Predicted price is

$$\hat{y} = 13.4 + 113.1 \times 2.2 = 262.3.$$



We used `lm` function to fit the linear model for the housing data. This function uses least squares loss function to estimate the parameters of the line. One of the nice properties of the least squares estimator is that it has a closed-form solution. This means that we can find the values of the parameters that minimize the loss function by solving a system of linear equations rather than using an optimization algorithm. The linear system is obtained by taking the gradient (multivariate derivative) of the loss function with respect to the parameters and setting it to zero. The loss function

$$\mathcal{L}(\beta) = \sum_{i=1}^n (y_i - f_\beta(x_i))^2$$

is a quadratic function of the parameters, so the solution is unique and can be found analytically. The gradient of the loss function with respect to the parameters is

$$\nabla \mathcal{L}(\beta) = -2 \sum_{i=1}^n (y_i - f_\beta(x_i)) \nabla f_\beta(x_i).$$

Given that $f_\beta(x_i) = \beta^T \psi(x_i)$, the gradient of the loss function with respect to the parameters is

$$\nabla \mathcal{L}(\beta) = -2 \sum_{i=1}^n (y_i - \beta^T \psi(x_i)) \psi(x_i)^T.$$

Setting the gradient to zero gives us the normal equations

$$\sum_{i=1}^n y_i \psi(x_i)^T = \sum_{i=1}^n \beta^T \psi(x_i) \psi(x_i)^T.$$

In matrix form, the normal equations are

$$\Psi^T y = \Psi^T \Psi \beta \quad (12.1)$$

where Ψ is the design matrix with rows $\psi(x_i)^T$ and y is the vector of output values. The solution to the normal equations is

$$\hat{\beta} = (\Psi^T \Psi)^{-1} \Psi^T y.$$

```
y <- d$price
Psi <- cbind(1, d$livingArea)
lhs <- t(Psi) %*% Psi
rhs <- t(Psi) %*% y
beta <- solve(lhs, rhs)
beta[, 1]
## 13 113
```

The function `solve` solves the Equation 12.1, and indeed finds the same values as the `lm` function. Essentially this is what the `lm` function does under the hood. The `solve` function uses the elimination method to solve the system of linear equations. The method we all learned when we were introduced to linear equations. The technique is known in linear algebra as LU decomposition.

In our housing example we used a linear model to predict the price of a house based on its square footage. The model is simple and easy to interpret, making it suitable for both prediction and interpretation. The model provides insights into the relationship between house size and price, allowing us to understand how changes in house size affect the price. The model can also be used to make accurate predictions of house prices based on square footage. This demonstrates the versatility of linear models for both prediction and interpretation tasks.

To further demonstrate the versatility of linear models, let's consider an example that allows us to understand the relationship between the performance of a stock portfolio managed by John Maynard Keynes and overall market performance.

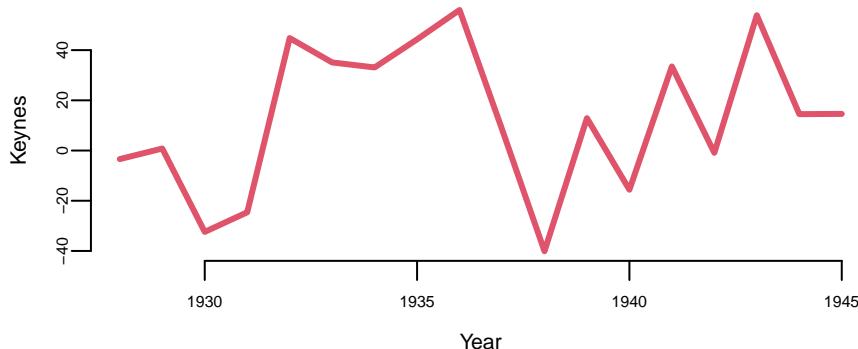
Example 12.2 (Keynes Investment Performance). John Maynard Keynes was a good investor, achieving a long-term average annual return of 16% while managing the King's College Cambridge endowment fund from 1921 to 1946. This performance significantly outperformed the overall market during that period. In the 1921-1929 period he experimented with investments in art

and currency, as well as cyclical equity investing. However, later in 1930s and onwards Keynes transitioned to a value investing strategy, focusing on undervalued stocks with strong fundamentals. This coincided with the Great Depression, where such investments offered significant opportunities. This shift led to consistent outperformance, with the King's College endowment generating returns exceeding the market on average. In our analysis we consider his returns during the 1928-1945 period. Below is the plot of his returns.

```
keynes <- read.csv("../data/keynes.csv", header = T)
```

Year	Keynes	Market	Rate
1928	-3.4	7.9	4.2
1929	0.8	6.6	5.3
1930	-32.4	-20.3	2.5
1931	-24.6	-25.0	3.6
1932	44.8	-5.8	1.5
1933	35.1	21.5	0.6

```
attach(keynes)
# Plot the data
plot(Year, Keynes, pch = 20, col = 34, cex = 3, type = "l",
      lwd = 3)
```



```
mean(Keynes)
## 13
```

Let's compare his performance to the overall stock market. The Dow Jones Industrial Average, grew at an average annual rate of 8.5% during the same

period (1921-1946). Keynes was able to consistently generate alpha, exceeding the market's overall returns.

The blue dots on Figure 12.1 below shows the relationship scatterplot of the market returns vs Keynes portfolio returns. The correlation coefficient is 0.76, which is high; when markets are doing well, Keynes also did well.

Now we fit a linear regression model

$$\text{Keynes} = \alpha + \beta \text{Market}$$

Note that here instead of generic notations β_0 for intercept and β_1 for slope, we use α and β , which are common in finance literature.

```
plot(Market, Keynes, xlab = "Market Return", ylab = "Keynes
      Excess Return", col = 20, bg = "lightblue", pch = 21, cex =
      0.8)
model <- lm(Keynes ~ Market)
abline(model, lwd = 3)
```

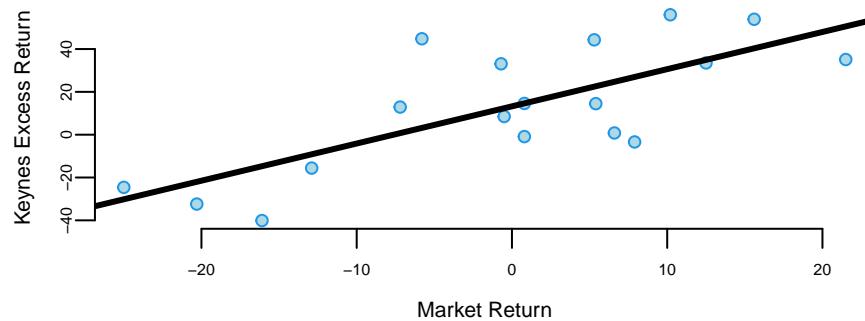


Figure 12.1: Scatterplot of the market returns vs Keynes portfolio returns.

```
coef(model)
## (Intercept)      Market
##           13.2          1.7
model %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	13.2	4.75	2.8	0.01

term	estimate	std.error	statistic	p.value
Market	1.7	0.39	4.5	0.00

The intercept of the least squares line is $\alpha = 13.2$, which is significantly higher than 0. This indicates that Keynes was able to generate higher returns than the market, even when the market was performing well. This is consistent with his value investing strategy, which allowed him to identify undervalued stocks and generate significant alpha.

Now we adjust for the risk-free returns and calculate excess return.

```
Keynes <- Keynes - Rate
Market <- Market - Rate
modelnew <- lm(Keynes ~ Market)
modelnew %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	14.5	4.69	3.1	0.01
Market	1.8	0.37	4.8	0.00

We see that after the adjustment, the intercept $\alpha = 14.46$ is now even larger.

12.1 Statistical Properties of Linear Models

Previously we demonstrated how the Central Limit Theorem enables us to derive key statistical properties of the sample mean, including its asymptotic normality, unbiasedness, and the relationship between sample size and estimation precision. These same principles extend naturally to linear models, where we can derive analogous properties for the least squares estimators of regression coefficients.

First, we introduce a regression model using the language of probability. A regression model assumes that the mean of the output variable y depends linearly on predictors x_1, \dots, x_p

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p + \epsilon, \text{ where } \epsilon \sim N(0, \sigma^2).$$

Often, we use simpler dot-product notation

$$y = \beta^T x + \epsilon,$$

where $\beta = (\beta_0, \beta_1, \dots, \beta_p)$ is the vector of regression coefficients and $x = (1, x_1, \dots, x_p)$ is the vector of predictors, with 1 appended to the beginning.

A more convenient form of the model is as follows

$$y | \beta, x \stackrel{iid}{\sim} N(\beta^T x, \sigma^2).$$

The additional term ϵ is a random variable that captures the uncertainty in the relationship between y and x ; it is called the error term or the residual. The error term is assumed to be normally distributed with mean zero and variance σ^2 . Thus, the linear regression model has a new parameter σ^2 that models dispersion of y_i around the mean $\beta^T x$, let's see an example.

12.1.1 Estimates and Intervals

In our housing example, we estimated the parameter β_1 to be equal to 113.12 and made a conclusion that the price of the house goes up by that amount when the living area goes up by one unit. However, the estimated value is based on a sample. The sample is a result of well-designed data collection procedure and is representative of the population, and we should expect the estimated value to be close to the true value. However, the estimated value is not the true value of the parameter, but an estimate of it. The true value of the parameter is unknown and the estimated value is subject to sampling error.

The sampling error is modeled by the normal distribution. The standard error of the estimate is a measure of the uncertainty in the estimated value of the parameter. The standard error is calculated from the sample data and is used to calculate confidence intervals and p-values for the estimated parameter.

We used the `lm` function to estimate the parameters of the linear model. The estimated values of the parameters are given in the `Estimate` column of the output. The estimated value of the intercept is $\hat{\beta}_0 = 13.439394$ and the estimated value of the slope is $\hat{\beta}_1 = 113.1225418$. These values are calculated using the least squares loss function, which minimizes the sum of squared differences between the predicted and actual values of the output variable. The estimated values of the parameters are subject to sampling error, which is modeled by the normal distribution. The standard error of the estimates is given in the `Std. Error` column of the output. The standard error is a measure of the uncertainty in the estimated values of the parameters. The t-statistic is the ratio of the estimated coefficient to its standard error. The p-value is the probability of observing a value at least as extreme as the one

observed, assuming the null hypothesis is true. In this case, the p-value for the `livingArea` coefficient is less than 0.05, so we conclude that the coefficient is statistically significant. This means that the size of the house is a statistically significant predictor of the price. The Residual standard error is the standard deviation of the residuals $\hat{y}_i - y_i$, $i = 1, \dots, n$.

Example 12.3 (House Prices). Let's go back to the Saratoga Houses dataset

term	estimate	std.error	statistic	p.value
(Intercept)	13	5.0	2.7	0.01
livingArea	113	2.7	42.2	0.00

The output of the `lm` function has several components. Besides calculating the estimated values of the coefficients, given in the `Estimate` column, the method also calculates standard error (`Std. Error`) and t-statistic (`t value`) for each coefficient. The t-statistic is the ratio of the estimated coefficient to its standard error. The p-value (`Pr(>|t|)`) is the probability of observing a value at least as extreme as the one observed, assuming the null hypothesis is true. The null hypothesis is that the coefficient is equal to zero. If the p-value is less than 0.05, we typically reject the null hypothesis and conclude that the coefficient is statistically significant. In this case, the p-value for the `livingArea` coefficient is less than 0.05, so we conclude that the coefficient is statistically significant. This means that the size of the house is a statistically significant predictor of the price. The Residual standard error is the standard deviation of the residuals $\hat{y}_i - y_i$, $i = 1, \dots, n$.

The estimated values of the parameters were calculated using least squares loss function discussed above. The residual standard error is also relatively easy to calculate from the model residuals

$$s_e = \sqrt{\frac{1}{n-2} \sum_{i=1}^n (\hat{y}_i - y_i)^2}.$$

Now the question is, how was the p-value for the estimates calculated? And why did we assume that ϵ is normally distributed in the first place? The normality of ϵ and as a consequence, the conditional normality of $y | x \stackrel{iid}{\sim} N(\beta^T x, \sigma^2)$ is easy to explain; it is simply due to mathematical convenience. Plus, this assumption happens to describe the reality well in a wide range of applications. One of those conveniences is our ability to calculate the mean and variance of the distribution of $\hat{\beta}_0$ and $\hat{\beta}_1$.

To understand how to calculate the p-values, we first notice that there is uncertainty about the values of the parameters β s. To get a feeling for the amount of variability in our experiments, imagine that we have two sample

data sets. For example, we have housing data from two different realtor firms. Do you think that the estimated value of price per square foot will be the same for both of those? The answer is no. Let's demonstrate with an example; we simulate 20 data sets from the same distribution and estimate 20 different linear models.

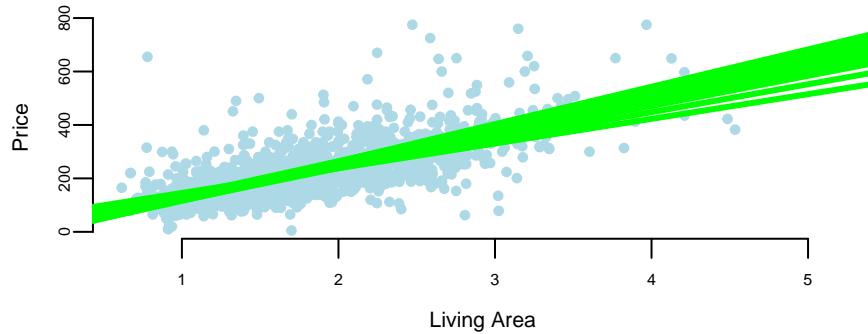


Figure 12.2: Twenty different linear models estimated using randomly selected subsample of the data.

Figure 12.2 shows the results of this simulation. We can see that the estimated coefficients $\hat{\beta}_i$ are different for each of the 20 samples. This is due to the sampling error. The sampling error is the difference between the estimated value of a parameter and its true value. The value of β_1 will differ from sample to sample. In other words, it will be a random variable. The sampling distribution of β_1 describes how it varies over different samples with the x values fixed. Statistical view of linear regression allows us to calculate confidence and prediction intervals for estimated parameters. It turns out that when least squares principle is used, the estimated $\hat{\beta}_1$ is normally distributed: $\hat{\beta}_1 | \beta_1 \sim N(\beta_1, s_1^2)$. Let's see how we can derive this result.

The extension of the central limit theorem, sometimes called the Lindeberg CLT, states that a linear combination of independent random variables that satisfy some mild condition are approximately normally distributed. We can show that estimates of $(\beta_0, \dots, \beta_p)$ are linear combinations of the observed values of y and are therefore normally distributed. Indeed, if we write the linear regression model in matrix form

$$y = X\beta + \epsilon,$$

where Y is the vector of observed values of the dependent variable, X is the matrix of observed values of the independent variables, β is the vector of unknown parameters, and ϵ is the vector of errors. Then, if we take the derivative of the loss function for linear regression and set it to zero, we get

the following expression for the estimated parameters

$$\hat{\beta} = Ay, \hat{y} = X\hat{\beta} = Hy.$$

where $A = (X^T X)^{-1} X^T$, $H = XA = X(X^T X)^{-1} X^T$ is the hat matrix. Due to Lindeberg central limit theorem, $\hat{\beta}$ is normally distributed. This is a useful property that allows us to calculate confidence intervals and p-values for the estimated parameters.

Now, we need to compute the mean and variance of $\hat{\beta}$. The mean is easy to compute, since the expectation of the sum is the sum of expectations, we have $\hat{\beta} = A(X\beta + \epsilon)$, hence

$$\hat{\beta} = \beta + A\epsilon$$

The expectation and variance of $\hat{\beta}$ are then:

$$\begin{aligned} E(\hat{\beta}) &= E(\beta + A\epsilon) = E(\beta) + E(A\epsilon) = \beta \\ \text{Var}(\hat{\beta}) &= \text{Var}(A\epsilon) = A\text{Var}(\epsilon)A^T = \sigma^2 A(X^T X)^{-1} A^T = \sigma^2 (X^T X)^{-1} \end{aligned}$$

Putting together the expectation and variance, we get the following distribution for $\hat{\beta}$:

$$\hat{\beta} \sim N(\beta, \sigma^2 (X^T X)^{-1}).$$

We can think of the estimated coefficients $\hat{\beta}_i$ as an average amount of change in y , when x_i goes up by one unit. Since this average was calculated using a sample data, it is subject to sampling error and the sampling error is modeled by the normal distribution. Assuming that residuals ϵ are independently normally distributed with a variance that does not depend on x (homoscedasticity), we can calculate the mean and variance of the distribution of $\hat{\beta}_i$. This is a useful property that allows us to calculate confidence intervals and p-values for the estimated parameters.

In summary, the statistical view of the linear regression model is useful for understanding the uncertainty associated with the estimated parameters. It also allows us to calculate confidence intervals and prediction intervals for the output variable.

1. Average value of output y is a linear function of input x and lie on the straight line of regression $\hat{y}_i = \beta^T x_i$.
2. The values of y are statistically independent.
3. The true value of $y = \hat{y}_i + \epsilon_i$ is a random variable, and it is normally distributed around the mean with variance σ^2 . This variance is the same for all values of y .
4. The estimated values of the parameters $\hat{\beta}_i$ are calculated from observed data and are subject to the sampling error and we are not certain about them. This uncertainty is modeled by the normally

distributed around the true values β . Given that errors ϵ_i are homoscedastic and independent, we have $Var(\hat{\beta}) = \sigma^2(X^T X)^{-1}$.

Again, consider a house example. Say in our data we have 10 houses with the same square footage, say 2000. Now the third point states, that the prices of those houses should follow a normal distribution and if we are to compare prices of 2000 sqft houses and 2500 sqft houses, they will have the same standard deviation. The second point means that price of one house does not depend on the price of another house.

All of the assumptions in the regression model can be written using probabilist notations:

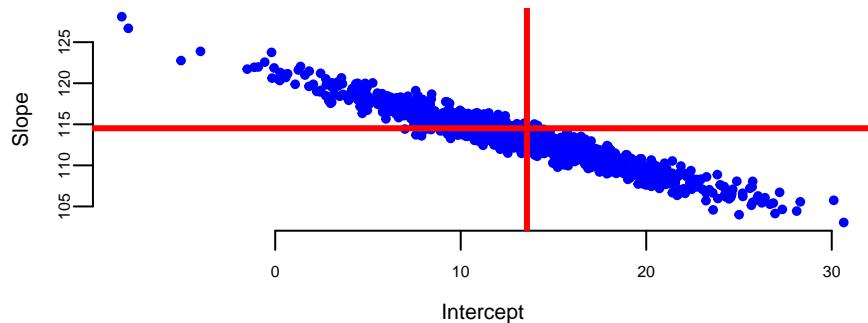
$$y | x \stackrel{iid}{\sim} N(\beta^T x, \sigma^2).$$

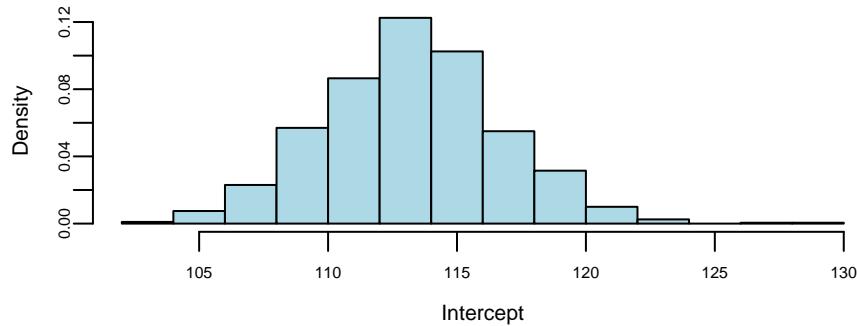
In the case when we have only one predictor the variance of the estimated slope $\hat{\beta}_1$ is given by

$$\text{Var}(\hat{\beta}_1) = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{\sigma^2}{(n-1)s_x^2},$$

where s_x^2 is the sample variance of x . Thus, there are three factors that impact the size of standard error for β_1 : sample size (n), error variance (σ^2), and x -spread, s_x .

We can empirically demonstrate the sampling error by simulating several samples from the same distribution and estimating several linear models. We can see that the estimated coefficients $\hat{\beta}_i$ are normally distributed around the true values β_i . If we plot coefficients for 1000 different models, we can see that the empirical distribution resembles a normal distribution.





Accounting for uncertainty in $\hat{\beta}$ s we can calculate confidence intervals for the predicted average \hat{y} . When we additionally account for the uncertainty in the predicted value \hat{y} , we can calculate prediction intervals.

Another advantage of adopting a statistical view of the linear regression model is ability to quantify information about potential outliers. Outliers are points that are extreme relative to our model predictions. Recall, that the residual is $e_i = y_i - \hat{y}_i$. Since our predicted value \hat{y}_i follows a normal distribution, the residual also follows a normal distribution, since it is a difference of normal random variable \hat{y}_i and a constant y_i . It easy to see that

$$e_i \sim N(0, s_e^2),$$

where s_e^2 is an empirical estimate of the error's variance.

Consider the relation between the *fitted values* \hat{y}_i and residuals e_i . Our predictions are given by the line. The residual e_i and predicted value \hat{y}_i for the i th observation are related via

$$y_i = \hat{y}_i + (y_i - \hat{y}_i) = \hat{y}_i + e_i.$$

Residuals allow us to define outliers. They simply have large residuals. We re-scale the residuals by their standard errors. This lets us define

$$r_i = \frac{e_i}{s_e} = \frac{y_i - \hat{y}_i}{s_e}$$

Since residuals follow normal distribution $e \sim N(0, \sigma^2)$, in 95% of the time we expect the standardized residuals to satisfy $-2 < r_i < 2$. Any observation with $|r_i| > 3$ is an extreme outlier; it is three sigmas away from the mean.

Another type of observations we are interested in are the *influential points*. These are observations that affect the magnitude of our estimates $\hat{\beta}$ s. They are important to find as they typically have economic consequences. We will use Cook's distance to assess the significance of an influential point. Cook's

distance associated with sample i measures the change in estimated model parameters $\hat{\beta}$ when sample i is removed from the training data set.

Intuitively, we model **regression-back-to-the-mean** effect. This is one of the most interesting statistical effects you'll see in daily life. In statistics, regression does not mean "going backwards", but rather the tendency for a variable that is extremely high or low to move closer to the average upon subsequent measurement. For example, Francis Galton, who was a cousin of Charles Darwin, in his study on regression to the mean height showed that if your parents are taller than the average, you'll regress back to the average. While people might expect the children of tall parents to be even taller and the children of short parents to be even shorter, Galton found that this wasn't the case. Instead, he observed that the heights of the children tended to be closer to the average height for the population. Galton termed this phenomenon "regression towards mediocrity" (now more commonly known as "regression to the mean"). It meant that extreme characteristics (in this case, height) in parents were likely to be less extreme (closer to the average) in their children. It is a classic example that helped introduce and explain this statistical concept. Galton's finding was one of the first insights into what is now a well-known statistical phenomenon. It doesn't imply that all individual cases will follow this pattern; rather, it's a trend observed across a population. It's important to understand that regression to the mean doesn't suggest that extreme traits diminish over generations but rather that an extreme measurement is partly due to random variation and is likely to be less extreme upon subsequent measurement.

Another example was documented by Daniel Kahneman and Amos Tversky in their book Thinking, Fast and Slow. They found that when a person performs a task, their performance is partly due to skill and partly due to luck. They observed that when a person performs a task and achieves an extreme result, their subsequent performance is likely to be less extreme. Particularly they studied effect of criticism and praise used by Israeli Air Force fighter pilots trainers. After criticism, the low-scoring pilots were retested. Often, their scores improve. At first glance, this seems like a clear effect of feedback from the trainer. However, some of this improvement is likely a statistical artifact and demonstrates the regression to the mean effect.

Why? Those pilots who initially scored poorly were, statistically speaking, somewhat unlucky. Their low scores may have been due to a bad day, stress, or other factors. When retested, their scores are likely to be closer to their true skill level, which is closer to the average. This natural movement towards the average can give the illusion that the intervention (praise or criticism) was more effective than it actually was. Conversely, if the top performers were praised and retested, we might find their scores decrease slightly, not necessarily due to the inefficacy of the praise but due to their initial high scores being partly due to good luck or an exceptionally good day. In conclusion, in pilot training and other fields, it's important to consider regression to the mean

when evaluating the effectiveness of interventions. Without this consideration, one might draw incorrect conclusions about the impact of training or other changes.

Example 12.4 (Google vs S&P 500). We will demonstrate how we can use statistical properties of a linear regression model to understand the relationship between returns of a google stock and the S&P 500 index. We will use Capital Asset Pricing Model (CAPM) regression model to estimate the expected return of an investment into Google stock and to price the risk. The CAPM model is

$$\text{GOOG} = \alpha + \beta \text{SP500} + \epsilon$$

On the left hand side, we have the return that investors expect to earn from investing into Google stock. In the CAPM model, this return is typically modeled as a dependent variable.

The input variable `SP500` represents the average return of the entire US market. Beta measures the volatility or systematic risk of a security or a portfolio in comparison to the market as a whole. A beta greater than 1 indicates that the security is more volatile than the market, while a beta less than 1 indicates it is less volatile. Alpha is the intercept of the regression line, it measures the excess return of the security over the market. The error term ϵ captures the uncertainty in the relationship between the returns of Google stock and the market.

In a CAPM regression analysis, the goal is to find out how well the model explains the returns of a security based on its beta. This involves regressing the security's excess returns (returns over the risk-free rate) against the excess returns of the market. The slope of the regression line represents the beta, and the intercept should ideally be close to the risk-free rate, although in practice it often deviates. This model helps in understanding the relationship between the expected return and the systematic risk of an investment.

Based on the uncertainty associated with the estimates for alpha and beta, we can formulate several hypothesis tests, for example:

Hypothesis	Question
$H_0 : \beta = 0$	Is Google related to the market?
$H_0 : \alpha = 0$	Does Google outperform the market in a consistent fashion?

```
getSymbols(Symbols = c("GOOG", "SPY"), from = "2017-01-03", to =
  "2023-12-29")
## "GOOG" "SPY"
```

```

gret <- as.numeric(dailyReturn(GOOG))
spyret <- as.numeric(dailyReturn(SPY))
l <- lm(gret ~ spyret)
tidy(l) %>% knitr::kable(digits = 4)

```

term	estimate	std.error	statistic	p.value
(Intercept)	0.0003	0.0003	1.1	0.27
spyret	1.1706	0.0240	48.8	0.00

Google vs S&P 500 returns between 2017-2023

```

plot(gret, spyret, bg = "lightblue", xlab = "Google Return",
      ylab = "SPY Return")
abline(l, lwd = 3, col = "red")

```

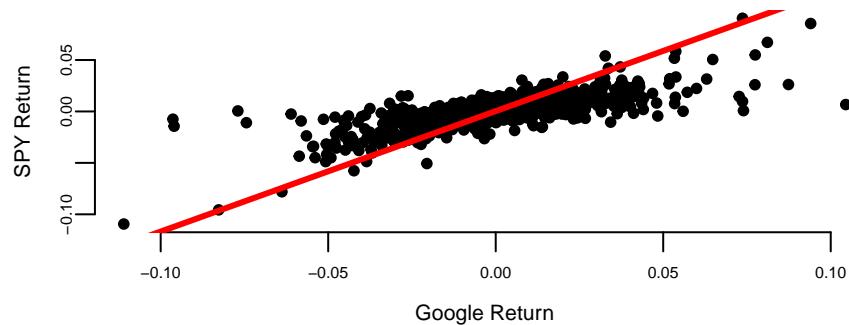


Figure 12.3: Google vs S&P 500 returns between 2017-2023

Here's what we get after we fit the model using `lm` function

Our best estimates are

$$\hat{\alpha} = 0.0004, \hat{\beta} = 1.01$$

Now we can provide the results for the hypotheses we set at the beginning. Given that the p-value for $H_0 : \beta = 0$ is $<2e-16$ we can reject the null hypothesis and conclude that Google is related to the market. The p-value for $H_0 : \alpha = 0$ is 0.06, which is greater than 0.05, so we cannot reject the null hypothesis and conclude that Google does not outperform the market in a consistent fashion in the 2017-2023 period.

Further, we can answer some of the other important questions, such as how much will Google move if the market goes up 10%?

```

alpha <- coef(l)[1]
beta <- coef(l)[2]
# Calculate the expected return
alpha + beta * 0.1
## (Intercept)
##          0.12

```

However, if we look at the earlier period between 2005-2016 (the earlier days of Google) the results will be different.

```

getSymbols(Symbols = c("GOOG", "SPY"), from = "2005-01-03", to =
  "2016-12-29")
## "GOOG" "SPY"
gret <- as.numeric(dailyReturn(GOOG))
spyret <- as.numeric(dailyReturn(SPY))
l <- lm(gret ~ spyret)
tidy(l) %>% knitr::kable(digits = 4)

```

Table 12.7: Google vs S&P 500 returns between 2005-2016

term	estimate	std.error	statistic	p.value
(Intercept)	0.0006	0.0003	2.2	0.03
spyret	0.9231	0.0230	40.1	0.00

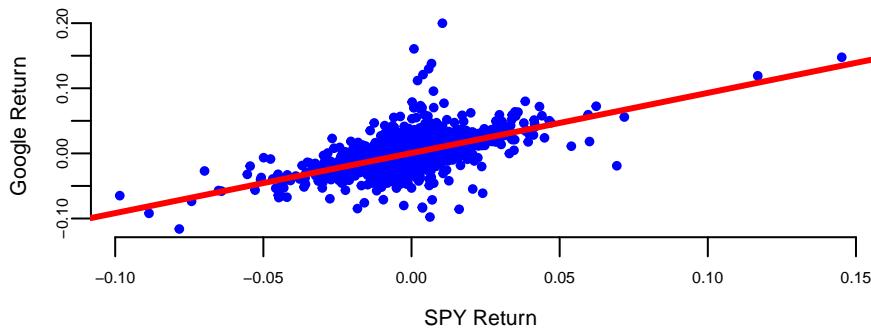


Figure 12.4: Google vs S&P 500 returns between 2005-2016

In this period Google did consistently outperform the market. The p-value for $H_0 : \alpha = 0$ is 0.03.

Example 12.5 (CAPM Model for Yahoo! Stock). Rather than estimate μ directly, the CAPM estimates the difference between μ and the risk-free rate r_f . This quantity $\mu - r_f$ is known as the expected excess return (excess relative to a risk-free investment). The CAPM relates the expected excess return of a stock to that of an underlying benchmark, typically a broad-based market index. Let μ_M and σ_M denote the return and volatility on the market index. The implication of CAPM is that there is a linear relationship between the expected excess return of a stock, $\mu - r_f$, and the excess return of the market, $\mu_M - r_f$.

$$\begin{aligned} \text{Excess Return}_{\text{Stock}} &= \beta \text{ Excess Return}_{\text{Market}} \\ \mu - r_f &= \beta(\mu_M - r_f) \end{aligned}$$

Put simply, the expected excess return of a stock is β times the excess expected return of the market. Beta (β) is a measure of a stock's risk in relation to the market. A beta of 1.3 implies that the excess return on the stock is expected to move up or down 30% more than the market. A beta bigger than one implies the stock is riskier than the market and goes up (and down) faster than the market goes up (and down). A beta less than one implies the stock is less risky than the market.

Using the CAPM, the expected return of the stock can now be defined as the risk free interest rate plus beta times the expected excess return of the market,

$$\mu = \text{Expected Return}_{\text{Stock}} = r_f + \beta(\mu_M - r_f)$$

Beta is typically estimated from a regression of the individual stock's returns on those of the market. The other parameters are typically measured as the historical average return on the market μ_M and the yield on Treasury Bills r_f . Together these form an estimate of μ . The volatility parameter σ is estimated by the standard deviation of historical returns.

Our qualitative discussion implicitly took the year as the unit of time. For our example, we make one minor change and consider daily returns so that μ and σ are interpreted as a daily rate of return and daily volatility (or standard deviation). We use an annual risk-free rate of 5%; this makes a daily risk-free rate of .019%, $r_f = 0.00019$, assuming there are 252 trading days in a year. A simple historical average is used to estimate the market return (μ_M) for the Nasdaq 100. The average annual return is about 23%, with corresponding daily mean $\mu_M = 0.00083$. A regression using daily returns from 1996-2000 leads to an estimate of $\beta = 1.38$. Combining these (pieces) leads to an estimated expected return of Yahoo!, $\mu_{\text{Yahoo!}} = 0.00019 + 1.38(0.00083 - 0.00019) = 0.00107$ on a daily basis. Note that the CAPM model estimates a future return that is much lower than the observed rate over the last three-plus years of .42% per day or 289% per year.

To measure the riskiness of Yahoo! notice that the daily historical volatility is 5%, i.e. $\sigma = 0.05$. On an annual basis this implies a volatility of $\sigma\sqrt{T} =$

$0.05\sqrt{252} = 0.79$, that is 79%. For comparison, the benchmark Nasdaq 100 has historical daily volatility 1.9% and an annual historical volatility of 30%. The estimates of all the parameters are recorded in Table 12.8.

Table 12.8: Key Parameter Estimates Based on Daily Returns 1996–2000

Asset	Expected return	Volatility	Regression coefft (s.e.)
Yahoo!	$\mu = 0.00107$	$\sigma = 0.050$	$\beta = 1.38(.07)$
Nasdaq 100	$\mu_M = 0.00083$	$\sigma_M = 0.019$	1
Treasury	$r_f = 0.00019$	—	—
Bills			

12.2 Factor Regression and Feature Engineering

A linear model assumes that output variable is proportional to the input variable plus an offset. However, this is not always the case. Often, we need to transform input variables by combining multiple inputs into a single predictor, for example by taking a ratio or putting inputs on a different scale, e.g. log-scale. In machine learning, this process is called feature engineering.

One of the classic examples of feature engineering is Fama-French three-factor model which is used in asset pricing and portfolio management. The model states that asset returns depend on (1) market risk, (2) the outperforming of small versus big companies, and (3) the outperformance of high book/market versus small book/market companies, mathematically

$$r = R_f + \beta(R_m - R_f) + b_s \cdot SMB + b_v \cdot HML + \alpha$$

Here R_f is risk-free return, R_m is the return of market, SMB stands for "Small market capitalization Minus Big" and HML for "High book-to-market ratio Minus Low"; they measure the historic excess returns of small caps over big caps and of value stocks over growth stocks. These factors are calculated with combinations of portfolios composed by ranked stocks (BtM ranking, Cap ranking) and available historical market data.

12.2.1 Logarithmic and Power Transformations

Consider, the growth of the Apple stock between 2000 and 2024. With the exception of the 2008 financial crisis period and 2020 COVID 19 related declines, the stock price has been growing exponentially. Figure 12.5 shows the

price of the Apple stock between 2000 and 2024. The price is closely related to the company's growth.

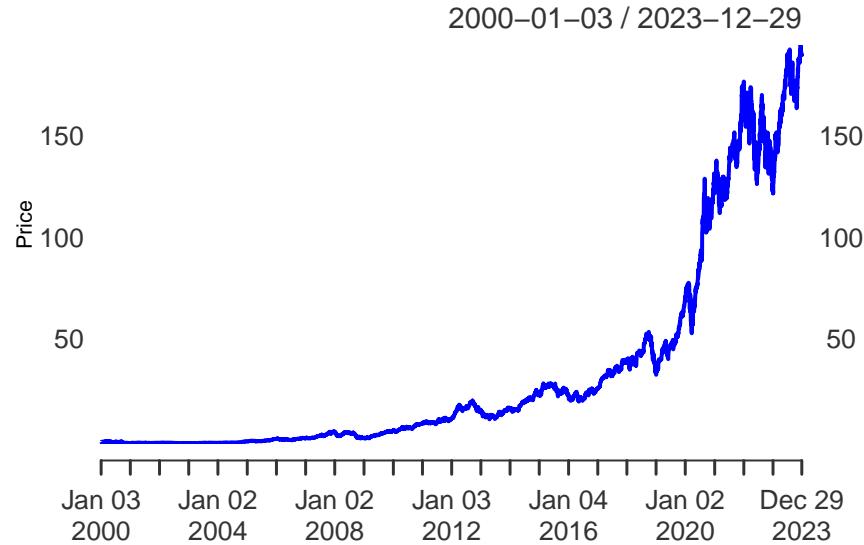


Figure 12.5: Apple stock price growth in the 2000-2024 period

The 2008 and 2020 declines are more related to extraneous factors, rather than the growth of the company. Thus, we can conclude that the overall growth of the company is exponential. The growth of a successful company typically follows the rule of compounding. Compounding is a fundamental principle that describes how some quantity grows over time when this quantity increases by a fixed percentage periodically. This is a very common phenomenon in nature and business. For example, if two parents have 2.2 children on average, then the population increases by 10% every generation. Another example is growth of investment in a savings account.

A more intuitive example is probably an investment in a savings account. If you invest 1000 in a savings account with 10% annual interest rate and you get paid once a year, then your account value will be 1100 by the end of the year. However, if you get paid n times a year, and initially invest w_0 , the final value w_t of the account after t years will be

$$w_t = w_0 \times (1 + r/n)^{nt}$$

where r is the annual interest rate. When you get paid every month ($n = 12$), a traditional payout schedule used by banks, then

$$w_t = 1000 \times (1 + 0.1/12)^{12} = 1105.$$

A value slightly higher than the annual payout of 1100.

The effect of compounding is minimal in the short term. However, the effect of compounding is more pronounced when the growth rate is higher and time periods are longer. For example at $r = 2$, $n = 365$ and 4-year period $t = 4$, you get

$$w_t = 1000 \times (1 + 2/365)^{3 \times 365} = 2,916,565.$$

Your account is close to 3 million dollars! Compared to $n = 1$ scenario

$$w_t = 1000 \times (1 + 2)^4 = 81,000,$$

when you will end up with merely 81 thousand. This is why compounding is often referred to as the “eighth wonder of the world” in investing contexts, emphasizing its power in growing wealth over time.

In general, as n goes up, the growth rate of the quantity approaches the constant

```
T <- 1:100
r <- 1
plot(T, (1 + r / T)^T, type = "l", col = "blue", ylab = "Future
      Value")
abline(h = exp(r), col = "red", lwd = 3)
```

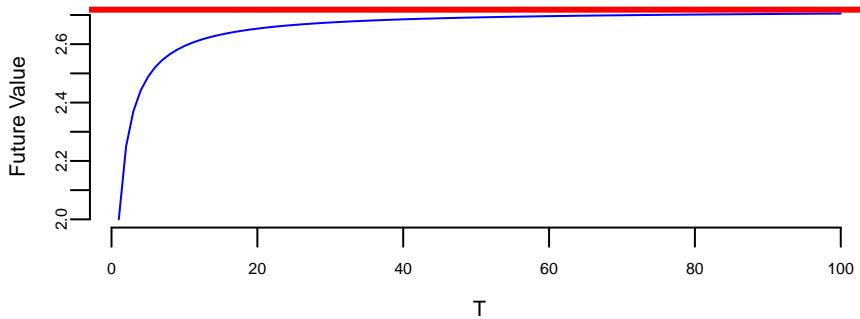


Figure 12.6: Growth of an investment in a savings account when n increases and return rate is 100% per year

Figure 12.6 shows the growth of an investment in a savings account when n increases and return rate is 100% per year. We can see that the growth rate approaches the constant $e \approx 2.72$ as n increases.

$$(1 + r/n)^n \rightarrow e^r, \text{ as } n \rightarrow \infty.$$

This limit was first delivered by Leonhard Euler and the number e is known as Euler's number.

Coming back to the growth of the Apple company, we can think of it growing at a small constant rate every day. The relation between the time and size of Apple is multiplicative. Meaning when time increases by one day, the size of the company increases by a small constant percentage. This is a multiplicative relation. In contrast, linear relation is additive, meaning that when time increases by one day, the size of the company increases by a constant amount. The exponential growth model is given by the formula

$$y = y_0 \times e^{\beta^T x}.$$

There are many business and natural science examples where multiplicative relation holds. If we apply the log function to both sides of the equation, we get

$$\log y = \log y_0 + \beta^T x.$$

This is a linear relation between $\log y$ and x . Thus, we can use linear regression to estimate the parameters of the exponential growth model by putting the output variable y on the log-scale.

Another example of nonlinear relation that can be analyzed using linear regression is when variables are related via a power law. This concept helps us model proportional relationships or ratios. In a multiplicative relationship, when one variable changes on a percent scale, the other changes in a directly proportional manner, as long as the multiplying factor remains constant. For example, the relation between the size of a city and the number of cars registered in the city is given by a power law. When the size of the city doubles, the number of cars registered in the city is also expected to double. The power law model is given by the formula

$$y = \beta_0 x^{\beta_1}.$$

If we apply the log function to both sides of the equation, we get

$$\log y = \log \beta_0 + \beta_1 \log x.$$

This is a linear relation between $\log y$ and $\log x$. Thus, we can use linear regression to estimate the parameters of the power law model by putting the output variable y and input variable x on the log-scale.

However, there are several caveats when putting variables on the log-scale. We need to make sure that the variable is positive. This means that we cannot apply log transformations to variables that contain zeros, such as dummy variables or count variables with zero values.

Example 12.6 (World's Smartest Mammal). We will demonstrate the power relation using data on brain (measured in grams) and body (measured in

kilograms) weights for 62 mammal species. The data was collected by Harry J. Jerison in 1973. The dataset contains the following variables:

Mammal	Brain	Body
African_elephant	5712.0	6654.00
African_giant_pouched_rat	6.6	1.00
Arctic_Fox	44.5	3.38
Arctic_ground_squirrel	5.7	0.92
Asian_elephant	4603.0	2547.00
Baboon	179.5	10.55

Let's build a linear model.

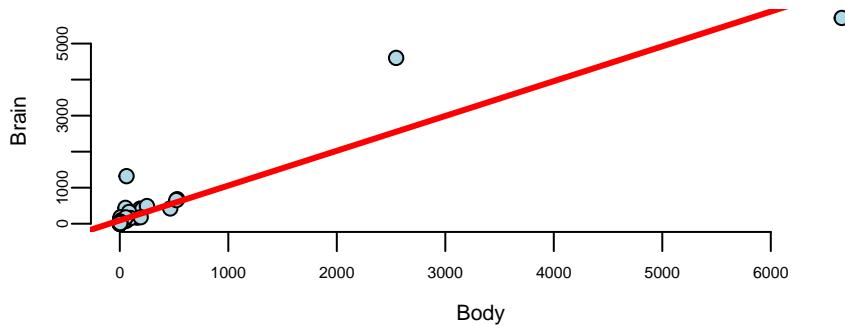


Figure 12.7: Brain vs Body weight for 62 mammal species

We see a few outliers with large residuals. This suggests that normality assumption is violated. We can check the residuals by plotting residuals against fitted values and plotting fitted vs true values.

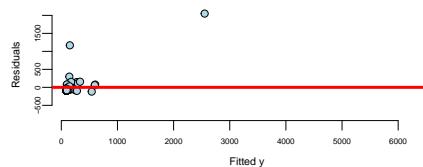


Figure 12.8: Fitted y vs Residuals

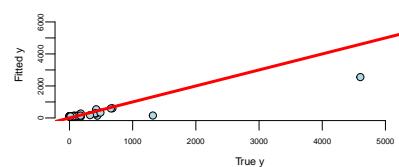


Figure 12.9: Fitted y vs True y

Remember, that residuals should roughly follow a normal distribution with mean zero and constant variance. We can see that the residuals are not normally distributed and the variance increases with the fitted values. This is

a clear indication that we need to transform the data. We can try a log-log transformation.

term	estimate	std.error	statistic	p.value
(Intercept)	2.18	0.11	20	0
log(mammals\$Body)	0.74	0.03	23	0

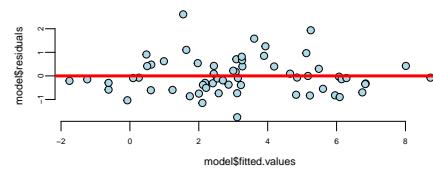


Figure 12.10: Fitted y vs Residuals

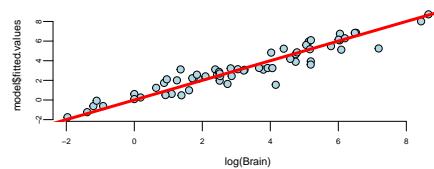


Figure 12.11: Fitted y vs True y

That is much better! The residuals variance is constant and the plot of fitted vs true values shows a linear relationship. The log-log model is given by the formula

$$\log \text{Brain} = 2.18 + 0.74 \log \text{Body}.$$

seem to achieve two important goals, namely linearity and constant variance. The coefficients are highly significant.

Although the log-log model fits the data rather well, there are a couple of outliers there. Let us print the observations with the largest residuals.

	Mammal	Brain	Body	Std.Res	Residual	Fit
11	Chinchilla	64	0.42	3.41	2.61	4.7
34	Man	1320	62.00	2.53	1.93	190.7
50	Rhesus_monkey	179	6.80	2.06	1.58	36.9
6	Baboon	180	10.55	1.64	1.26	51.1
42	Owl_monkey	16	0.48	1.44	1.10	5.1
10	Chimpanzee	440	52.16	1.26	0.96	167.7
27	Ground_squirrel	4	0.10	1.18	0.91	1.6
43	Patas_monkey	115	10.00	1.11	0.85	49.1
60	Vervet	58	4.19	1.06	0.81	25.7
3	Arctic_Fox	44	3.38	0.92	0.71	22.0

There are two outliers, the Chinchilla and the Human, both have disproportionately large brains!

In fact, the Chinchilla has the largest standardized residual of 3.41. Meaning that the predicted value of 4.7 g is 3.41 standard deviations away from the

recorded value of 64 g. This suggests that the Chinchilla is an intellectually superior species! However, after checking more carefully we realized that there was a recording error and the actual weight of an average Chinchilla's brain is 6.4. We mistyped the decimal separator! Thus the actual residual is 0.4.

```
abs(model$fitted.values[11] - log(6.4)) / sd(model$residuals)
## 11
## 0.4
```

In reality Chinchilla's brain is not far from an average mammal of this size!

Example 12.7 (Newfood). A six month market test has been performed on the Newfood product, which is a breakfast cereal. The goal is to build a multiple regression model that provides accurate sales forecasts. This dataset represents the outcome of a controlled experiment in which the values of the independent variables that affect sales were carefully chosen by the analyst.

The analysis aims to identify the factors that contribute to sales of a new breakfast cereal and to quantify the effects of business decisions such as the choice of advertising level, location in store, and pricing strategies.

variable	description
<code>sales</code>	new cereal sales
<code>price</code>	price
<code>adv</code>	low or high advertising (0 or 1)
<code>locat</code>	bread or breakfast section (0 or 1)
<code>inc</code>	neighborhood income
<code>svol</code>	size of store

First, we need to understand which variables need to be transformed. We start by running the “kitchen-sink” regression with all variables. Then we perform diagnostic checks to assess model assumptions and identify potential issues. Based on these diagnostics, we decide which variables should be transformed. After running the new model with transformations, we perform additional diagnostics and variable selection to refine the model. Using the final model after transformations and eliminating variables, we examine what the largest Cook’s distance is to identify influential observations. Finally, we provide a summary of coefficients and their statistical significance.

First, let’s examine the correlation matrix to understand the relationships between all variables in the dataset. This will help us identify potential multicollinearity issues and understand the strength and direction of associations between variables before building our regression model.

```
newfood <- read.csv("../data/newfood.csv")
attach(newfood)
names(newfood)
## "sales" "price" "adv"     "locat"   "income" "svol"    "city"
#> "indx"
# knitr::kable()
head(newfood)
```

	sales	price	adv	locat	income	svol	city	indx
225	24	0	0	7.3	34	3	1	
190	24	0	0	7.3	34	3	2	
205	24	0	0	7.3	34	3	3	
323	24	0	0	8.3	41	4	1	
210	24	0	0	8.3	41	4	2	
241	24	0	0	8.3	41	4	3	

```
# correlation matrix
cm <- cor(cbind(sales, price, adv, locat, income, svol))
cm[upper.tri(cm, diag = TRUE)] <- NA
# knitr::kable(as.table(round(cm, 3)))
as.table(round(cm, 3)) %>% knitr::kable()
```

	sales	price	adv	locat	income	svol
sales	NA	NA	NA	NA	NA	NA
price	-0.66	NA	NA	NA	NA	NA
adv	0.00	0.00	NA	NA	NA	NA
locat	0.00	0.00	0.00	NA	NA	NA
income	0.16	-0.13	-0.75	0.00	NA	NA
svol	0.38	-0.18	-0.74	-0.04	0.81	NA

Remember, correlations between variables are not the same as regression coefficients (β 's)! Looking at the correlation matrix, we can see that total sales volume (**svol**) is negatively correlated with advertising (**adv**), and income (**income**) is also negatively correlated with advertising (**adv**). The question is how might these negative correlations impact our ability to estimate the true advertising effects in our regression model?

```
as.table(round(cm[2:4, 1:3], 3)) %>% knitr::kable()
```

	sales	price	adv
price	-0.66	NA	NA
adv	0.00	0	NA
locat	0.00	0	0

There's no correlation in the X 's by design! Let's start by only including `price`, `adv`, `locat`

```
model <- lm(sales ~ price + adv + locat)
model %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	562.31	53.1	10.58	0.00
price	-12.81	1.8	-7.20	0.00
adv	0.22	14.5	0.02	0.99
locat	-0.22	14.5	-0.02	0.99

Why is the marketer likely to be upset by this regression? Why is the economist happy? The marketer might be upset because the advertising coefficient looks small or insignificant due to omitted variable bias (e.g. not controlling for store volume or income properly). The economist is happy because the price coefficient is negative, consistent with the law of demand. Let's add `income` and `svol` to the regression and use log-log model.

```
model <- lm(log(sales) ~ log(price) + adv + locat +
  log(income) + log(svol))
model %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	8.41	1.39	6.06	0.00
log(price)	-1.74	0.22	-7.90	0.00
adv	0.15	0.10	1.49	0.14
locat	0.00	0.06	0.02	0.99
log(income)	-0.52	0.50	-1.06	0.29
log(svol)	1.03	0.26	4.04	0.00

Why no logs for `adv` and `locat` variables? The $\log(svol)$ coefficient is close to one!

The reason we don't apply logarithms to `adv` and `locat` variables is because they are binary categorical variables (taking values 0 or 1). Taking the logarithm of 0 is undefined, and taking the logarithm of 1 equals 0, which would not provide any meaningful transformation. For binary variables, the exponential transformation in the final model interpretation directly gives us the multiplicative effect on sales when the variable changes from 0 to 1.

Regarding the $\log(svol)$ coefficient being close to one (1.03), this suggests that sales scale approximately proportionally with store volume. A coefficient of 1.0 would indicate perfect proportional scaling, meaning a 1% increase in store volume would lead to a 1% increase in sales. Our coefficient of 1.03 indicates slightly more than proportional scaling—a 1% increase in store volume leads to a 1.03% increase in sales, suggesting some economies of scale or network effects in larger stores.

On the transformed scale (log-log model),

$$\log \text{sales} = 8.41 - 1.74 \log \text{price} + 0.150 \text{adv} + 0.001 \text{locat} - 0.524 \log \text{inc} + 1.03 \log \text{svol}$$

On the un-transformed scale,

$$\text{sales} = e^{8.41} (\text{price})^{-1.74} e^{0.15 \text{adv}} e^{0.001 \text{locat}} (\text{inc})^{-0.524} (\text{svol})^{1.03}$$

In the log-log regression model, the relationship between sales and the continuous variables (price, income, and store volume) follows a power function relationship. This means that a 1% change in these variables leads to a proportional change in sales according to their respective coefficients. Specifically, a 1% increase in price leads to a 1.74% decrease in sales, a 1% increase in income leads to a 0.524% decrease in sales, and a 1% increase in store volume leads to a 1.03% increase in sales.

In contrast, the binary variables (advertising and location) follow an exponential relationship with sales. When advertising is present (`adv=1`), sales increase by a factor of $e^{0.15} = 1.16$, representing a 16% improvement. Similarly, when a store is in a good location (`locat=1`), sales increase by a factor of $e^{0.001} = 1.001$, representing a 0.1% improvement. This exponential relationship arises because these variables are binary (0 or 1) and cannot be log-transformed, so their effects are multiplicative on the original sales scale.

The log-log regression model reveals several important relationships between the independent variables and sales performance.

- Price elasticity is $\hat{\beta}_{\text{price}} = -1.74$. A 1% increase in price will drop sales 1.74%
- $\text{adv} = 1$ increases `sales` by a factor of $e^{0.15} = 1.16$. That's a 16% improvement

We should delete the `locat` variable from our regression model because it is statistically insignificant. The coefficient for `locat` has a very small magnitude (0.001) and a high p-value, indicating that there is insufficient evidence to reject the null hypothesis that this variable has no effect on sales. Including statistically insignificant variables in a model can lead to overfitting and reduce the model's predictive accuracy on new data. By removing `locat`, we create a more parsimonious model that focuses only on the variables that have meaningful relationships with the outcome variable.

Now, we are ready to use our model for prediction. `predict.lm` provides a \hat{Y} -prediction given a new X_f

```
modelnew <- lm(log(sales) ~ log(price) + adv + log(income) +
  ↵ log(svol))
modelnew %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	8.41	1.37	6.1	0.00
log(price)	-1.74	0.22	-8.0	0.00
adv	0.15	0.10	1.5	0.14
log(income)	-0.52	0.49	-1.1	0.29
log(svol)	1.03	0.25	4.1	0.00

```
newdata <- data.frame(price = 30, adv = 1, income = 8, svol =
  ↵ 34)
predict.lm(modelnew, newdata, interval = "confidence", level =
  ↵ 0.99)
##   fit lwr upr
## 1 5.2 4.9 5.5
```

Exponentiate-back to find sales = $e^{5.1739} = 176.60$.

12.3 Interactions

In many situations, x_1 and x_2 interact when predicting y . An interaction occurs when the effect of one independent variable on the dependent variable

changes at different levels of another independent variable. For example, consider a study analyzing the effect of study hours x_1 and a tutoring program x_2 , a binary variable where 0 = no tutoring, 1 = tutoring) on test scores y . Without an interaction term, we assume the effect of study hours on test scores is the same regardless of tutoring. With an interaction term, we can explore whether the effect of study hours on test scores is different for those who receive tutoring compared to those who do not. Here are a few more examples when there is potential interaction.

Examples of potential interactions include whether gender changes the effect of education on wages, whether patients recover faster when taking drug A, and how advertisement affects price sensitivity. Interactions are particularly useful with dummy variables. We can build a kitchen-sink model with all possible dummies (day of the week, gender, etc.).

If we think that the effect of x_1 on y depends on the value of x_2 , we model it using a linear relation

$$\beta_1 = \beta_{10} + \beta_{11}x_2$$

and the model without interaction $y = \beta_0 + \beta_1x_1 + \beta_2x_2$ becomes

$$y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \beta_3x_1x_2 + \epsilon.$$

The interaction term captures the effect of x_1 on y when $x_2 = 1$. The coefficient β_3 is the difference in the effect of x_1 on y when $x_2 = 1$ and $x_2 = 0$. If β_3 is significant, then there is an interaction effect. If β_3 is not significant, then there is no interaction effect. The coefficients β_1 and β_2 are called marginal effects.

In R:

```
model <- lm(y = x1 * x2)
```

estimates $y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \beta_3x_1x_2 + \epsilon$, and

```
model <- lm(y = x1:x2)
```

estimates only $y = \beta_3x_1x_2 + \epsilon$

If β_3 is significant there's an interaction effect and we must leave β_1 and β_2 in the model whether they are significant or not.

When $x_2 = D$ is a dummy variable with values of zero or one, we typically run a regression of the form

$$y = \beta_0 + \beta_1x_1 + \beta_2x_1 \star D + \epsilon$$

The coefficient $\beta_1 + \beta_2$ is the effect of x_1 when $D = 1$. The coefficient β_1 is the effect when $D = 0$.

Example 12.8 (Orange Juice). Understanding how advertising affects consumer price sensitivity is a fundamental question in marketing analytics. Do promotional campaigns make consumers more or less sensitive to price changes? To explore this question, we analyze a dataset from the orange juice category that includes sales data from 83 Chicagoland stores. This dataset provides rich information including price, sales volume (measured as log units moved), brand identity, and whether the product was featured in store displays or advertising circulars.

Let's begin by examining the structure of the data and the basic relationships between variables.

```
oj <- read.csv("./../data/obj.csv")
knitr::kable(oj[1:5, c(1:7, 10)], digits = 2)
```

store	brand	week	logmove	feat	price	AGE60	INCOME
2	tropicana	40	9.0	0	3.9	0.23	11
2	tropicana	46	8.7	0	3.9	0.23	11
2	tropicana	47	8.2	0	3.9	0.23	11
2	tropicana	48	9.0	0	3.9	0.23	11
2	tropicana	50	9.1	0	3.9	0.23	11

```
model <- lm(logmove ~ log(price) * feat, data = oj)
model %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

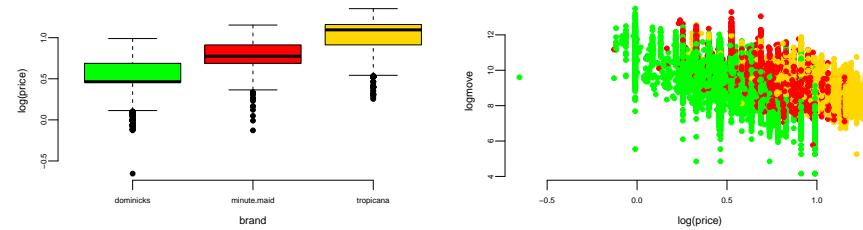
term	estimate	std.error	statistic	p.value
(Intercept)	9.66	0.02	588	0
log(price)	-0.96	0.02	-51	0
feat	1.71	0.03	56	0
log(price):feat	-0.98	0.04	-23	0

```
model <- lm(log(price) ~ brand - 1, data = oj)
model %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
branddominicks	0.53	0	254	0
brandminute.maid	0.79	0	382	0
brandtropicana	1.03	0	500	0

The dataset includes three major brands: Dominick's (the store brand), Minute Maid, and Tropicana. We can visualize the price distributions and the relationship between price and sales for these brands.

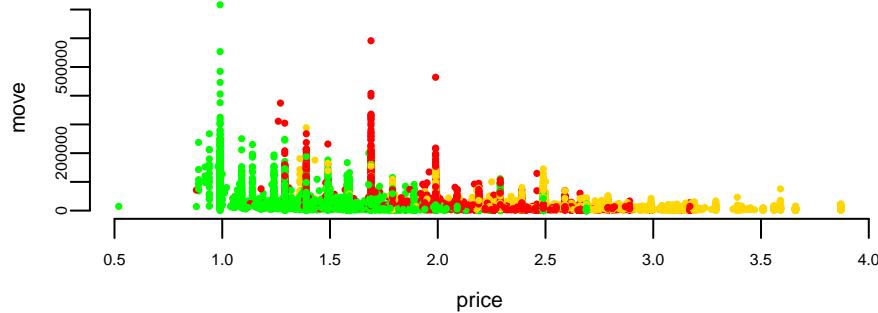
```
brandcol <- c("green", "red", "gold")
oj$brand <- factor(oj$brand)
boxplot(log(price) ~ brand, data = oj, col = brandcol)
plot(logmove ~ log(price), data = oj, col = brandcol[oj$brand],
  pch = 20)
```



The boxplot reveals that Tropicana commands premium pricing, while Dominick's store brand is positioned as the value option. The scatterplot shows a clear negative relationship between price and sales, as expected from basic economic theory: higher prices lead to lower sales volume.

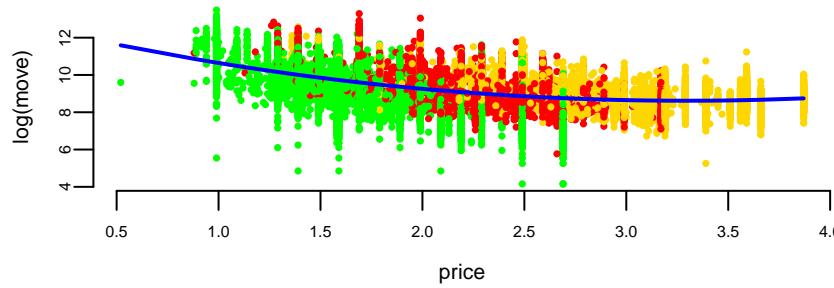
A critical modeling decision is whether to use logarithmic transformations for our variables. Let's examine the relationship between price and sales on both the original and log scales to determine which specification better captures the underlying relationship.

First, we plot sales against price on the original scale:



The pattern suggests a nonlinear relationship that might be better captured with a transformation. Next, we examine the relationship between price and $\log(\text{sales})$:

```
l1 <- loess(logmove ~ price, data = oj, span = 2)
smoothed1 <- predict(l1)
ind <- order(oj$price)
plot(logmove ~ price, data = oj, col = brandcol[oj$brand], pch =
  16, cex = 0.5, ylab = "log(move)")
lines(smoothed1[ind], x = oj$price[ind], col = "blue", lwd = 2)
```



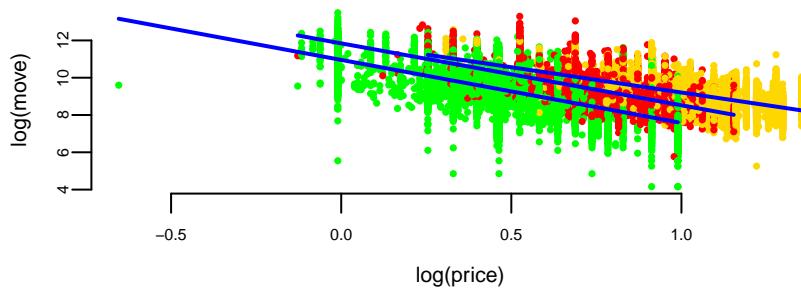
While this shows some improvement, the relationship still exhibits curvature. Finally, we examine the log-log specification:

```
par(mar = c(4, 4, 0.1, 0.1), bty = "n")
plot(logmove ~ log(price), data = oj, col = brandcol[oj$brand],
  pch = 16, cex = 0.5, ylab = "log(move)")
for (i in 1:3) {
  brand_data <- oj[oj$brand == levels(oj$brand)[i], ]
```

```

l2 <- lm(logmove ~ log(price), data = brand_data)
smoothed2 <- predict(l2)
ind <- order(brand_data$price)
lines(smoothed2[ind], x = log(brand_data$price[ind]), col =
  "blue", lwd = 2)
}

```



The log-log specification produces a much more linear relationship. This makes economic sense: the relationship between price and sales is multiplicative rather than additive. A 10% increase in price leads to a proportional percentage decrease in sales, regardless of the starting price level. This constant elasticity model is the standard approach in demand analysis.

Our central research question is whether featuring a product in advertisements affects consumers' price sensitivity. We start with a baseline model that includes both log price and the advertising indicator:

$$\log(\text{sales}) = \beta_0 + \beta_1 \log(\text{price}) + \beta_2 \text{feat}.$$

However, this model assumes that advertising only shifts demand up or down, without changing how consumers respond to price. To test whether advertising actually changes price sensitivity, we need to allow the price coefficient β_1 to vary with advertising. We can model this by assuming that the price sensitivity depends linearly on the advertising indicator:

$$\beta_1 = \beta_3 + \beta_4 \text{feat}.$$

Substituting this into our demand equation gives us an interaction model:

$$\log(\text{sales}) = \beta_0 + (\beta_3 + \beta_4 \text{feat}) \log(\text{price}) + \beta_2 \text{feat}.$$

Expanding this expression yields:

$$\log(\text{sales}) = \beta_0 + \beta_3 \log(\text{price}) + \beta_4 \text{feat} \times \log(\text{price}) + \beta_2 \text{feat}.$$

The coefficient β_4 on the interaction term tells us how much price sensitivity changes when the product is featured in advertising. Let's estimate this model and compare it to simpler specifications.

First, we fit the basic model with only log price:

```
lm(logmove ~ log(price), data = obj) %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	10.4	0.02	679	0
log(price)	-1.6	0.02	-87	0

This simple model suggests a price elasticity of approximately -3.1, meaning a 1% increase in price leads to roughly a 3.1% decrease in sales.

Next, we add brand effects and the advertising indicator without interactions:

```
lm(logmove ~ log(price) + feat + brand, data = obj) %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	10.28	0.01	708	0
log(price)	-2.53	0.02	-116	0
feat	0.89	0.01	85	0
brandminute.maid	0.68	0.01	58	0
brandtropicana	1.30	0.01	88	0

Now we estimate the full model with the interaction between price and advertising:

```
objreg <- lm(logmove ~ log(price) * feat, data = obj)
objreg %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

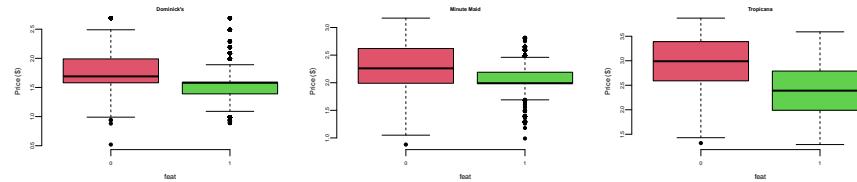
term	estimate	std.error	statistic	p.value
(Intercept)	9.66	0.02	588	0
log(price)	-0.96	0.02	-51	0
feat	1.71	0.03	56	0
log(price):feat	-0.98	0.04	-23	0

The results reveal a striking finding: the interaction coefficient is -0.98, which is highly statistically significant. When products are not featured (`feat = 0`), the price elasticity is -0.96. However, when products are featured in advertising (`feat = 1`), the price elasticity becomes $-0.96 + (-0.98) = -1.94$, essentially doubling in magnitude. This means that advertising makes consumers nearly twice as sensitive to price changes.

This result seems counterintuitive at first. We might expect that advertising would make consumers less price-sensitive by building brand loyalty or highlighting non-price attributes. Why would advertising increase price sensitivity?

The explanation lies in the promotional strategy employed by retailers. Let's examine the pricing behavior during advertising campaigns:

```
doj <- oj %>% filter(brand == "dominicks")
boxplot(price ~ feat,
        data = oj[oj$brand == "dominicks", ], col = c(2, 3),
        main = "Dominick's", ylab = "Price ($)"
)
boxplot(price ~ feat,
        data = oj[oj$brand == "minute.maid", ], col = c(2, 3),
        main = "Minute Maid", ylab = "Price ($"
)
boxplot(price ~ feat,
        data = oj[oj$brand == "tropicana", ], col = c(2, 3),
        main = "Tropicana", ylab = "Price ($"
)
```



The boxplots reveal the key insight: prices are systematically lower when products are featured (`feat = 1`) compared to when they are not (`feat = 0`). This

is a classic promotional pricing strategy where advertising and price discounts are bundled together. Retailers use advertising to draw attention to temporary price reductions.

The increased price sensitivity we observe during advertising periods is not caused by the advertising itself making consumers more price-conscious. Rather, it reflects the fact that advertising campaigns coincide with periods of price variation, and consumers who respond to advertisements are precisely those who are most price-sensitive. The advertising serves as a signal that draws price-sensitive consumers into the market, creating a selected sample with higher elasticity.

This example demonstrates the importance of carefully interpreting interaction effects and understanding the underlying business practices that generate the data. What appears to be an advertising effect on price sensitivity is actually a selection effect driven by the correlation between advertising and pricing strategies.

12.4 Categorical Variables and Dummy Encoding

In our orange juice analysis, we want to understand how brand affects sales. Intuitively, we might want to include brand in our regression model:

$$\log(\text{sales}) = \beta_0 + \beta_1 \log(\text{price}) + \beta_2 \text{brand}$$

However, we immediately encounter a problem: brand is a categorical variable, not a numerical one. We have three brands in our dataset - Dominick's, Minute Maid, and Tropicana - but these are labels, not numbers. How can we incorporate such categorical information into a mathematical equation that requires numerical coefficients?

The solution lies in creating *dummy variables* (also called indicator variables). For a categorical variable with k categories, we create $k - 1$ binary variables, each taking the value 1 if the observation belongs to that category and 0 otherwise. One category is designated as the reference category and is represented when all dummy variables equal zero.

For our three-brand scenario, we create two dummy variables as shown in the table below:

Brand	Intercept	brandminute.maid	brandtropicana
minute.maid	1	1	0

Brand	Intercept	brandminute.maid	brandtropicana
tropicana	1	0	1
dominicks	1	0	0

This encoding transforms our regression equation into:

$$\log(\text{sales}) = \beta_0 + \beta_1 \log(\text{price}) + \beta_{21} \text{brandminute.maid} + \beta_{22} \text{brandtropicana}.$$

Fortunately, statistical software like R handles this transformation automatically when you include a categorical variable in your model specification:

```
lm(logmove ~ log(price) + brand, data = obj) %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	10.83	0.01	745	0
log(price)	-3.14	0.02	-137	0
brandminute.maid	0.87	0.01	67	0
brandtropicana	1.53	0.02	94	0

Example 12.9 (Golf Performance Data). Dave Pelz has written two best-selling books for golfers, *Dave Pelz's Short Game Bible*, and *Dave Pelz's Putting Bible*. These books have become essential reading for serious golfers looking to improve their performance through data-driven analysis and scientific methodology.

Dave Pelz was formerly a “rocket scientist” (literally) at NASA, where he worked on the Apollo space program. His background in physics and engineering provided him with the analytical skills to improve golf instruction through data analytics. His systematic approach to analyzing golf performance helped him refine his teaching methods and develop evidence-based strategies for improving players’ games. Through his research, Pelz discovered that it’s the short-game that matters most for overall scoring performance.

One of Pelz’s most famous findings concerns the optimal speed for a putt. Through extensive data collection and analysis, he determined that the best chance to make a putt is one that will leave the ball 17 inches past the hole, if it misses. This counterintuitive result challenges the common belief that golfers should aim to leave putts just short of the hole. Pelz’s research showed that putts hit with this specific speed have the highest probability of going

in, as they account for the natural variations in green speed, slope, and other factors that affect putt trajectory.

Now, we demonstrate how to use data to improve your golf game. We analyze the dataset that contains comprehensive year-end performance statistics for 195 professional golfers from the 2000 PGA Tour season. This rich dataset captures technical abilities of the players as well as financial success (measured by the amount of prize money they made). Each observation represents season's averages of the players' performance and total prize money. List below shows the variables in the dataset.

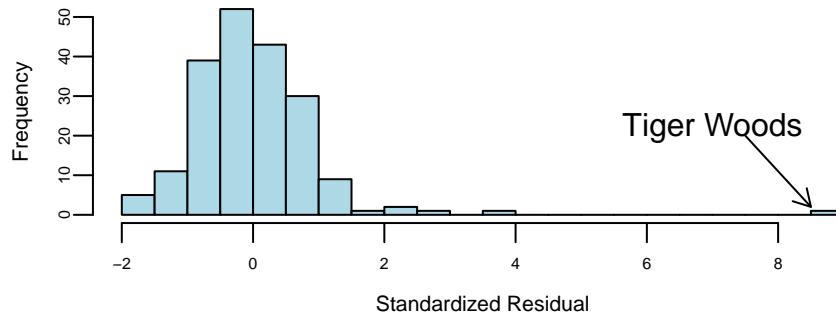
Variable	Description
<code>nevents</code>	The number of official PGA events included in the statistics
<code>money</code>	The official dollar winnings of the player
<code>drivedist</code>	The average number of yards driven on par 4 and par 5 holes
<code>gir</code>	Greens in regulation, measured as the percentage of time that the first (tee) shot on a par 3 hole ends up on the green, or the second shot on a par 4 hole ends up on the green, or the third shot on a par 5 hole ends up on the green
<code>avgputts</code>	The average number of putts per round

We will analyze these data to determine which of the variables `nevents`, `drivedist`, `gir`, and `avgputts` is most important for winning money on the PGA Tour. We begin by performing a regression of `Money` on all explanatory variables:

term	estimate	std.error	statistic	p.value
(Intercept)	14856638	4206466	3.5	0.00
<code>nevents</code>	-30067	11183	-2.7	0.01
<code>drivedist</code>	21310	6913	3.1	0.00
<code>gir</code>	120855	17429	6.9	0.00
<code>avgputts</code>	-15203045	2000905	-7.6	0.00

Let's look at the residuals:

```
hist(rstandard(model00),
  breaks = 20, col = "lightblue",
  xlab = "Standardized Residual", main = ""
)
```



It seems like we need to measure `money` on a log scale. Let's transform with `log(Money)` as it has much better residual diagnostic plots.

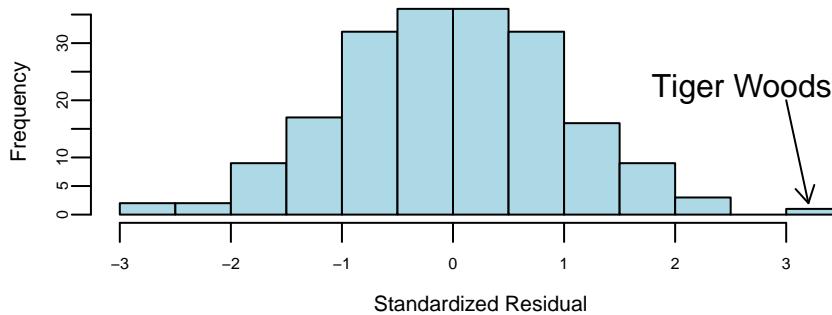
```
m <- lm(formula = log(money) ~ nevents + drivelist + gir +
  ↵ avgputts, data = d00)
m %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	36.15	3.58	10.10	0.00
nevents	-0.01	0.01	-0.94	0.35
drivelist	0.01	0.01	2.40	0.02
gir	0.17	0.01	11.18	0.00
avgputts	-21.13	1.70	-12.42	0.00

```
model00log <- lm(log(money) ~ nevents + drivelist + gir +
  ↵ avgputts, data = d00)
```

term	estimate	std.error	statistic	p.value
(Intercept)	36.15	3.58	10.10	0.00
nevents	-0.01	0.01	-0.94	0.35
drivelist	0.01	0.01	2.40	0.02
gir	0.17	0.01	11.18	0.00
avgputts	-21.13	1.70	-12.42	0.00

```
hist(rstandard(model00log),
  breaks = 20, col = "lightblue",
  xlab = "Standardized Residual", main = ""
)
arrows(x0 = 3, y0 = 20, x1 = 3.2, y1 = 2, length = 0.1)
text(x = 2.6, y = 22, labels = "Tiger Woods", cex = 1)
```



Using log scale for `money` gives us a better model. We will keep it for now. How about selecting variables. Notice, that t -stats for `nevents` is < 1.5 . Thus, we can remove it.

```
m1 <- lm(formula = log(money) ~ drivelist + gir + avgputts, data
          ↵   = d00)
m1 %>%
  tidy() %>%
  knitr::kable(digits = 2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	36.17	3.58	10.1	0.00
drivelist	0.01	0.01	2.5	0.01
gir	0.17	0.01	11.2	0.00
avgputts	-21.37	1.68	-12.7	0.00

It is obvious that fewer putts indicate a better golfer. However, decreasing the average number of putts per round by one is extremely difficult to achieve.

Evaluating the Coefficients

1. Greens in Regulation (GIR) has a $\hat{\beta} = 0.17$. If I can increase my GIR by one, I'll earn $e^{0.17} = 1.18\%$ An extra 18%.

2. DriveDis has a $\hat{\beta} = 0.014$. A 10 yard improvement, I'll earn $e^{0.014 \times 10} = e^{0.14} = 1.15\%$. An extra 15%.

Caveat: Everyone has gotten better since 2000!

Tiger Woods was nine standard deviations better than what the model predicted, while taking the natural logarithm of money earnings significantly improves the residual diagnostics and an exponential model appears to fit the data well as evidenced by the good residual diagnostic plots; furthermore, the t-ratios for the number of events variable are consistently under 1.5, indicating it may not be a significant predictor.

The outliers represent the biggest over and under-performers in terms of money winnings when compared with their performance statistics, and Tiger Woods, Phil Mickelson, and Ernie Els won major championships by performing exceptionally well during tournaments with substantial prize money available.

We can see the over-performers and under-performers in the data.

Table 12.32: Over-Performers

		name	money	Predicted	Error
1	Tiger Woods	9188321	3584241	5604080	
2	Phil Mickelson	4746457	2302171	2444286	
3	Ernie Els	3469405	1633468	1835937	
4	Hal Sutton	3061444	1445904	1615540	
20	Notah Begay III	1819323	426061	1393262	
182	Steve Hart	107949	-1186685	1294634	

Now, let's extract the list of underperformers, which are given by large negative residuals. According to our model, Glasson and Stankowski should win more money based on their performance statistics, but they are not achieving the expected earnings. This could be due to several factors: they might be performing well in practice rounds but struggling under tournament pressure, they could be playing in fewer high-payout events, or their performance metrics might not capture other important aspects of tournament success like clutch putting or mental toughness during critical moments.

Table 12.33: Under-Performers

		name	money	Predicted	Error
47	Fred Couples	990215	1978477	-988262	
52	Kenny Perry	889381	1965740	-1076359	
70	Paul Stankowski	669709	1808690	-1138981	
85	Bill Glasson	552795	1711530	-1158735	

	name	money	Predicted	Error
142	Jim McGovern	266647	1397818	-1131171

Lets look at 2018 data, the highest earners are

Table 12.34: Highest earners 2018

	name	nevents	money	drivedist	gir	avgputts
	Justin Thomas	23	8694821	312	69	1.7
	Dustin Johnson	20	8457352	314	71	1.7
	Justin Rose	18	8130678	304	70	1.7
	Bryson DeChambeau	26	8094489	306	70	1.8
	Brooks Koepka	17	7094047	313	68	1.8
	Bubba Watson	24	5793748	313	68	1.8

Overperformers

Table 12.35: Overperformers 2018

		name	money	Predicted	Error
1		Justin Thomas	8694821	5026220	3668601
2		Dustin Johnson	8457352	6126775	2330577
3		Justin Rose	8130678	4392812	3737866
4		Bryson DeChambeau	8094489	3250898	4843591
5		Brooks Koepka	7094047	4219781	2874266
6		Bubba Watson	5793748	3018004	2775744
9		Webb Simpson	5376417	2766988	2609429
11		Francesco Molinari	5065842	2634466	2431376
12		Patrick Reed	5006267	2038455	2967812
84		Satoshi Kodaira	1471462	-1141085	2612547

Underperformers

Table 12.36: Underperformers 2018

		name	money	Predicted	Error
102		Trey Mullinax	1184245	3250089	-2065844
120		J.T. Poston	940661	3241369	-2300708
135		Tom Lovelady	700783	2755854	-2055071
148		Michael Thompson	563972	2512330	-1948358

		name	money	Predicted	Error
150	Matt Jones	538681	2487139	-1948458	
158	Hunter Mahan	457337	2855898	-2398561	
168	Cameron Percy	387612	3021278	-2633666	
173	Ricky Barnes	340591	3053262	-2712671	
176	Brett Stegmaier	305607	2432494	-2126887	

Our analysis reveals three particularly interesting insights from the golf performance data. First, Tiger Woods demonstrates exceptional performance, appearing as an outlier eight standard deviations above the model's predictions. This indicates his extraordinary success relative to his statistical metrics. Second, the model shows that increasing driving distance by ten yards corresponds to a fifteen percent increase in earnings, suggesting that power off the tee provides a significant competitive advantage. Finally, improving greens in regulation (GIR) by one percentage point leads to an eighteen percent increase in earnings. This highlights the importance of approach shot accuracy in determining financial success on the PGA Tour. The model successfully identifies both under-performers and over-performers, providing valuable insights into which players may be exceeding or falling short of expectations based on their measurable skills.

So far, we have viewed regression coefficients β as fixed, unknown constants that we estimate from data. However, in many applications, we may have prior knowledge about these parameters or wish to handle uncertainty more formally. This motivates the Bayesian approach to regression.

12.5 Quantile Regression

While least squares regression provides an estimate of the conditional mean, it can be sensitive to outliers. In contrast, quantile regression offers a robust alternative by estimating conditional quantiles, such as the median.

The least absolute deviations (Quantile) loss function is the sum of absolute differences between the predicted and actual values. It is used for regression problems with continuous variables. The goal is to minimize the sum of absolute errors (SAE) to improve the predictive performance of the model. Given observed data set the least absolute deviations estimator is the value of β that minimizes the sum of absolute errors

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^n |y_i - f_{\beta}(x_i)|$$

The least absolute deviations estimator is also known as the quantile estimator, where the quantile is set to 0.5 (the median). This is because the least absolute deviations estimator is equivalent to the median of the data (the 0.5 quantile).

Again, in the unconditional case, when we do not observe any inputs x , the least absolute deviations estimator is the sample median. To solve this minimization problem, we need to consider the concept of a *subgradient*, since the absolute value function is not differentiable at zero.

A subgradient generalizes the idea of a derivative for functions that are not differentiable everywhere. For a convex function $f(x)$, a subgradient at a point x_0 is any value g such that for all x ,

$$f(x) \geq f(x_0) + g(x - x_0).$$

The set of all such g is called the *subdifferential* at x_0 .

For the absolute value function, the subgradient is given by:

$$\frac{\partial}{\partial x} |x| = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ [-1, 1] & \text{if } x = 0 \end{cases}$$

That is, at $x = 0$, any value between -1 and 1 is a valid subgradient.

Applying this to our minimization problem, the subgradient of the sum of absolute deviations with respect to β is:

$$\sum_{i=1}^n \text{sign}(y_i - \beta).$$

This equals to zero only when the number of positive items equals the number of negative which happens when β is the median.

A more rigorous and non-calculus proof is due to Schwertman, Gilks, and Cameron (1990). Let y_1, \dots, y_n be the observed data and $\hat{\beta}$ be the least absolute deviations estimator. Then we have

$$\sum_{i=1}^n |y_i - \hat{\beta}| \leq \sum_{i=1}^n |y_i - \beta|$$

for any β . Let $y_{(1)}, \dots, y_{(n)}$ be the ordered data. Then we have

$$\sum_{i=1}^n |y_i - \hat{\beta}| \leq \sum_{i=1}^n |y_i - y_{(i)}|$$

Let $y_{(n/2)}$ be the median of the data. Then we have

$$\sum_{i=1}^n |y_i - \hat{\beta}| \leq \sum_{i=1}^n |y_i - y_{(n/2)}|$$

which implies that $\hat{\beta}$ is the median of the data.

The generalization of the median estimator to the case of estimating value of quantile τ is as follows

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^n \rho_{\tau}(y_i - \beta)$$

where $\rho_{\tau}(x) = x(\tau - \mathbb{I}(x < 0))$ is the quantile loss function. If we set $\tau = 0.5$, the loss function becomes the absolute value function and we get the median estimator. The expected loss is

$$E\rho_{\tau}(y - \beta) = (\tau - 1) \int_{-\infty}^{\beta} (y - \beta) dF(y) + \tau \int_{\beta}^{\infty} (y - \beta) dF(y)$$

Differentiating the expected loss function with respect to β and setting it to zero gives the quantile estimator

$$\hat{\beta}_{\tau} = F^{-1}(\tau)$$

where F^{-1} is the quantile function of the distribution of y . Thus, the problem of finding a quantile is solved via optimization.

A key difference between the least squares and least absolute deviations estimators is their sensitivity to outliers. The least squares estimator is sensitive to outliers because it squares the errors, giving more weight to large errors. In contrast, the least absolute deviations estimator is less sensitive to outliers because it takes the absolute value of the errors, giving equal weight to all errors. This makes the least absolute deviations estimator more robust to outliers than the least squares estimator.

Another difference is the computational complexity. Least squares estimator can be found by solving a linear system of equations. There are fast and efficient algorithms for it, making the least squares estimator computationally efficient. In contrast, the least absolute deviations estimator requires more computationally expensive numerical optimization algorithms.

There is also a hybrid loss function, called *Huber loss*, which combines the advantages of squared errors and absolute deviations. It uses SE for small errors and AE for large errors, making it less sensitive to outliers.

12.6 Bayes Regression

The Bayesian approach to linear regression treats the regression coefficients as random variables rather than fixed unknowns. This perspective offers several

advantages: it provides a natural way to incorporate prior knowledge, quantifies uncertainty about parameter estimates through probability distributions, and allows for principled inference even with limited data.

Consider a linear regression model

$$f_\beta(x) = x^T \beta + \epsilon, \quad \epsilon \sim N(0, \sigma_e^2),$$

where $y = f_\beta(x)$ represents the observed response, x is a vector of predictors, and ϵ captures random noise with variance σ_e^2 . In the Bayesian framework, we specify a prior distribution over the coefficients

$$\beta \sim N(0, \Sigma).$$

The choice of prior covariance Σ encodes our beliefs about the likely values of β before seeing data. A diagonal $\Sigma = \sigma_\beta^2 I$ assumes independent, identically distributed coefficients with variance σ_β^2 . Smaller values of σ_β^2 express stronger beliefs that coefficients are close to zero, effectively imposing shrinkage similar to ridge regression.

The goal of Bayesian inference is to calculate the posterior distribution over model parameters given the observed data (X, y) :

$$p(\beta | X, y) = \frac{p(y | \beta, X)p(\beta)}{p(y | X)}.$$

The numerator contains the likelihood $p(y | \beta, X)$, which measures how well each β explains the data, and the prior $p(\beta)$, which encodes our initial beliefs. The denominator $p(y | X)$ is the marginal likelihood or evidence, which serves as a normalizing constant ensuring the posterior integrates to one.

A key property of the normal distribution is conjugacy: the product of two Gaussian density functions yields another Gaussian. This allows us to derive the posterior analytically:

$$\begin{aligned} p(\beta | X, y) &\propto \exp\left(-\frac{1}{2\sigma_e^2}(y - X^T \beta)^T (y - X^T \beta)\right) \exp\left(-\frac{1}{2}\beta^T \Sigma^{-1} \beta\right) \\ &\propto \exp\left(-\frac{1}{2}(\beta - \bar{\beta})^T A(\beta - \bar{\beta})\right) \end{aligned}$$

where we have completed the square in the exponent. This reveals that the posterior is Gaussian:

$$\beta | X, y \sim N(\bar{\beta}, A^{-1}),$$

where the precision matrix is

$$A = \sigma_e^{-2} X X^T + \Sigma^{-1},$$

and the posterior mean is

$$\bar{\beta} = \sigma_e^{-2} A^{-1} X y = A^{-1} (\sigma_e^{-2} X y).$$

The posterior mean $\bar{\beta}$ represents a compromise between the data and the prior. When we have abundant data (large n), the term $\sigma_e^{-2}XX^T$ dominates Σ^{-1} , and $\bar{\beta}$ approaches the ordinary least squares estimate. Conversely, with limited data, the prior exerts stronger influence, pulling $\bar{\beta}$ toward zero. The posterior covariance A^{-1} quantifies remaining uncertainty: diagonal entries give the variance of each coefficient, while off-diagonal entries capture correlations between parameters.

Example 12.10 (Posterior Distribution for Simple Linear Regression). To build intuition, consider a simple linear model with a single predictor ($p = 1$):

$$y = \beta_0 + \beta_1 x + \epsilon, \quad \beta_i \sim N(0, 1), \quad \sigma_e = 1.$$

The prior $\beta_i \sim N(0, 1)$ reflects moderate uncertainty: we expect coefficients to typically fall within $[-2, 2]$ (covering approximately 95% of probability mass), but we remain open to larger values if the data demand them.

Before observing any data, this prior induces a distribution over possible linear functions $y = \beta_0 + \beta_1 x$. Each draw (β_0, β_1) from the prior defines a different line. Sampling many such pairs generates an ensemble of lines representing our prior beliefs about plausible relationships between x and y . Let's visualize this:

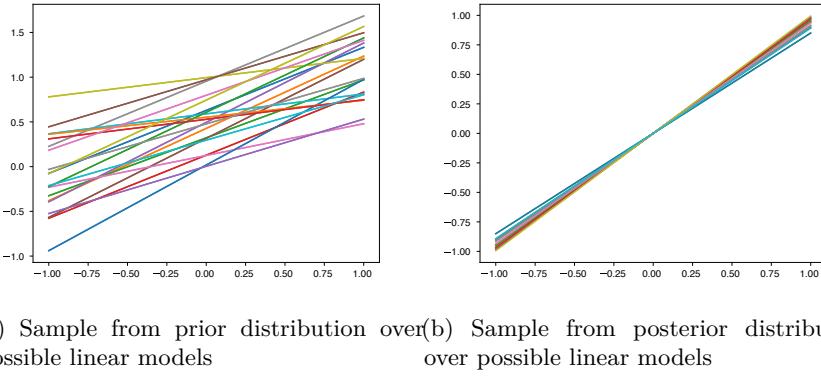


Figure 12.12: Bayesian linear regression: prior and posterior distributions over linear models

These visualizations illustrate the fundamental mechanism of Bayesian learning in linear regression. The left panel displays a sample of twenty linear functions drawn from the prior distribution $\beta_i \sim N(0, 1)$, representing our initial beliefs before observing any data. Notice the wide dispersion of lines with varying slopes and intercepts, reflecting substantial uncertainty about the true relationship. The right panel shows the same twenty functions after

conditioning on observed data points (shown as black dots). The posterior distribution dramatically concentrates around the data, with all sampled lines now passing close to the observations. This concentration demonstrates how Bayesian inference combines prior beliefs with empirical evidence to reduce uncertainty. The remaining spread in the posterior reflects both the inherent noise in the data ($\sigma_e = 1$) and residual parameter uncertainty. This shrinkage from a diffuse prior to a concentrated posterior exemplifies the core principle of Bayesian learning: data updates our beliefs in a principled, probabilistic manner.

12.6.1 Penalized Regression and Priors

In the standard linear regression setting, we minimize the sum of squared errors. However, when the number of predictors is large, or when we want to prevent overfitting, we often add a penalty term to the loss function. This is known as penalized regression or regularization.

The two most common forms of penalized regression are **Ridge Regression** and **Lasso Regression**.

- **Ridge Regression** adds a penalty proportional to the sum of squared coefficients ($\lambda \sum \beta_j^2$). This shrinks coefficients toward zero but rarely sets them exactly to zero.
- **Lasso Regression** adds a penalty proportional to the sum of absolute values of coefficients ($\lambda \sum |\beta_j|$). This has the unique property of setting some coefficients exactly to zero, effectively performing variable selection.

Interestingly, these frequentist regularization methods have direct Bayesian interpretations: * Ridge regression corresponds to placing a **Gaussian prior** on the coefficients: $\beta_j \sim N(0, \tau^2)$. * Lasso regression corresponds to placing a **Laplace prior** (Double Exponential) on the coefficients: $\beta_j \sim \text{Laplace}(0, b)$.

While Lasso is powerful for variable selection, it has a drawback: it shrinks all coefficients, including the large signals that we want to keep. This can introduce bias in our estimates of the important predictors. Ideally, we want a method that aggressively shrinks small coefficients (noise) to zero but leaves large coefficients (signals) relatively untouched.

12.6.2 Global-Local Priors and the Horseshoe

To address the limitations of Lasso, modern Bayesian statistics uses **Global-Local** shrinkage priors, which have two variance components for each coefficient: a global shrinkage parameter (τ) that pulls all coefficients toward zero (estimating the overall sparsity level), and a local shrinkage parameter (λ_j) for

each coefficient that allows specific signals to escape the global shrinkage. The **Horseshoe Prior** is a state-of-the-art global-local prior that uses half-Cauchy distributions for both parameters, encouraging coefficients to be either very close to zero (strong shrinkage) or very large (no shrinkage), making it nearly ideal for sparse signal recovery.

For a detailed theoretical treatment of the horseshoe prior, including its mathematical formulation, shrinkage properties, computational implementation, and comprehensive examples, see Section 17.10.



13

Logistic Regression and Generalized Linear Models

Classification predicts categories rather than numbers. Does this patient have the disease or not? Is this email spam? Is that object in the camera feed a pedestrian, a vehicle, or a traffic sign? The output is discrete—often just 0 or 1 (binary classification), sometimes multiple classes. Self-driving cars classify objects in real-time from camera feeds; medical systems flag high-risk patients; fraud detection systems sort transactions into suspicious and legitimate.

Given observed data $(x_i, y_i)_{i=1}^n$, where each y_i is either 0 or 1, we start by assuming a binomial likelihood function for the response variable, defined as follows:

$$P(y_i = 1 | p_i) = p_i^{y_i} (1 - p_i)^{1-y_i},$$

where p_i is the function of the inputs x_i and coefficients β that gives us the probability of the response variable taking on a value of 1, given the input variables. A typical approach to calculate p_i is to use the logistic function

$$\begin{aligned} f_\beta(x_i) &= \beta^T x_i \\ p_i &= \sigma(f_\beta(x_i)) = \frac{e^{f_\beta(x_i)}}{1 + e^{f_\beta(x_i)}}, \end{aligned}$$

where β is a vector of parameters. The logistic function $\sigma(\cdot)$ maps any real number to the interval $(0, 1)$, interpreting the output as a probability.



Figure 13.1: The Sigmoid Function

13.1 Model Fitting

Then we fit the model using binomial log-likelihood minimization. It leads us to the maximum likelihood estimator for parameters β (a.k.a *cross-entropy estimator*), defined as

$$\hat{\beta} = \arg \min_{\beta} \mathcal{L}(\beta),$$

where

$$\mathcal{L}(\beta) = - \sum_{i=1}^n [y_i \log p_i + (1 - y_i) \log (1 - p_i)].$$

Similar to the least squares estimator, the cross-entropy estimator optimization problem is convex, so it has a unique solution.

In the unconditional case (an intercept-only model with no inputs x), the cross-entropy estimator simplifies to the sample mean. If we take the derivative of the above expression with respect to β_0 and set it to zero, we get

$$-\frac{d}{d\beta_0} \sum_{i=1}^n [y_i \log (\beta_0) + (1 - y_i) \log (1 - \beta_0)] = -\sum_{i=1}^n \left[\frac{y_i}{\beta_0} - \frac{1 - y_i}{1 - \beta_0} \right] = 0$$

which gives us the solution

$$\hat{\beta}_0 = \frac{1}{n} \sum_{i=1}^n y_i.$$

which is the sample mean.

Unlike the least squares estimator or the unconditional case, the system of equations

$$\nabla \mathcal{L}(\beta) = 0$$

is not linear and cannot be solved by inverting a matrix. However, there are efficient iterative numerical optimization algorithms that can be used to find the optimal solution. The most common one is the *BFGS* (Broyden-Fletcher-Goldfarb-Shanno) algorithm. It is a quasi-Newton method that's particularly well-suited for optimizing the cross-entropy loss function in logistic regression.

When we have more than two classes $y \in \{1, \dots, K\}$, we build $K - 1$ models $f_{\beta_1}(x), \dots, f_{\beta_{K-1}}(x)$, one for each of the first $K - 1$ classes, while treating the K -th class as the reference class with $f_{\beta_K}(x) = 0$. We then use the softmax function to convert the outputs into probabilities:

For classes $j = 1, \dots, K - 1$:

$$P(y = j | x) = \frac{\exp(f_{\beta_j}(x))}{1 + \sum_{i=1}^{K-1} \exp(f_{\beta_i}(x))}$$

For the reference class K :

$$P(y = K \mid x) = \frac{1}{1 + \sum_{i=1}^{K-1} \exp(f_{\beta_i}(x))}$$

Some implementations of the logistic regression use K models, one for each class, and then use the softmax function to convert the outputs into probabilities. This is equivalent to the above approach, but it is more computationally expensive.

The vector of non-scaled outputs $(f_{\beta_1}(x), \dots, f_{\beta_{K-1}}(x))$ is called the *logits*.

The softmax function is a generalization of the logistic function to the case of more than two classes. It is often used as the activation function in the output layer of neural networks for multi-class classification problems. It converts the output of each model into a probability distribution over the classes, making it suitable for multi-class classification with probabilistic outputs.

The logistic function has a nice statistical interpretation. It is the CDF of the logistic distribution, which is a symmetric distribution with mean 0 and variance $\pi^2/3$, thus p_i is simply a value of this CDF, evaluated at $\beta^T x_i$.

Further, Logistic regression models the log-odds (logit) of the probability as a linear function of the predictors, which aligns with the maximum likelihood estimation framework and provides desirable statistical properties. Specifically, if we invert the logistic function,

$$p_i = \sigma(\beta^T x_i) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}},$$

we get the log-odds

$$\log\left(\frac{p_i}{1 - p_i}\right) = \beta^T x_i.$$

Meaning that $\beta^T x_i$ measures how probability of $y_i = 1$ changes with respect to the change in x_i on the log-odds scale. It allows us to interpret the model coefficients as the log-odds ratios of the response variable.

In some disciplines, such as econometrics, psychology and natural sciences, a normal CDF is used instead of the logistic CDF. This is often done for historical reasons or because the normal CDF implies slightly different theoretical assumptions that may be more appropriate for specific datasets.

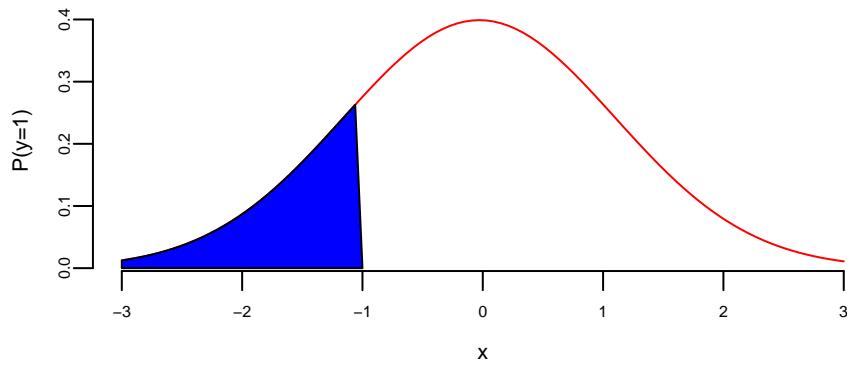
In the case of the normal CDF, the model is called *probit*, it stands for probability unit, and the link function is called *probit link*. The probit model is defined as

$$\Phi^{-1}(p_i) = \beta^T x_i.$$

where $\Phi(\cdot)$ is the normal CDF.

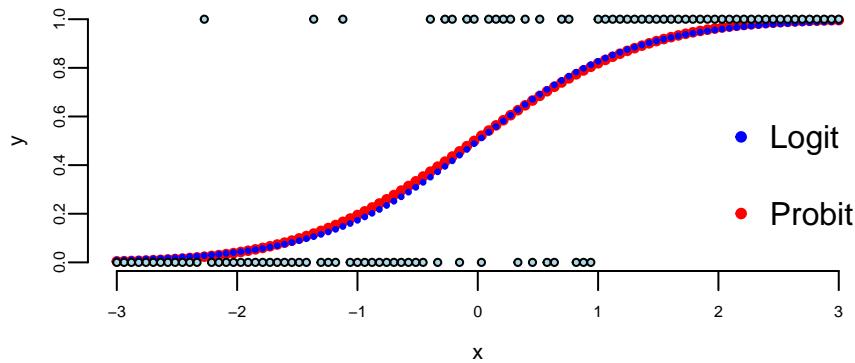
The term probit was coined in the 1930's by biologists studying the dosage-cure rate link. We can fit a probit model using `glm` function in R.

```
## -0.86
## 0.19
## 1
## 0.19
```



Our prediction is the blue area which is equal to 0.195.

Outside fields like behavioral economics, the logistic function is generally preferred over the probit model due to the interpretability of log-odds and its natural extension to multi-class problems. The PDF of the logistic distribution is very similar to the normal PDF.

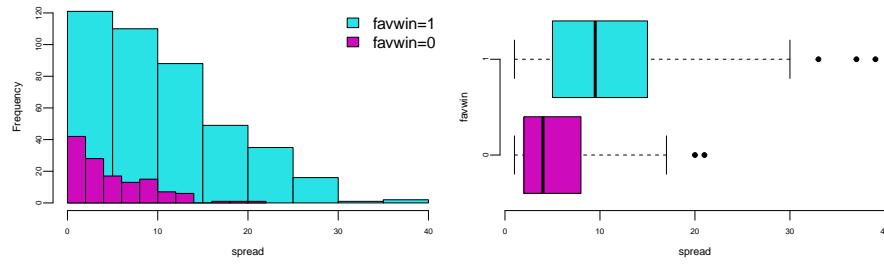


Example 13.1 (Example: NBA point spread). We will use the NBA point spread data to illustrate the logistic regression. The data is available in the `NBAspread.csv` file. The data contains the point spread for each game in the NBA from 2013 to 2014 season. The data also contains the outcome of the game, whether the favorite won or not. The point spread is the number of

points by which the favorite is expected to win the game and is predicted by the bookmakers. We simply want to see how well the point spread predicts the outcome of the game.

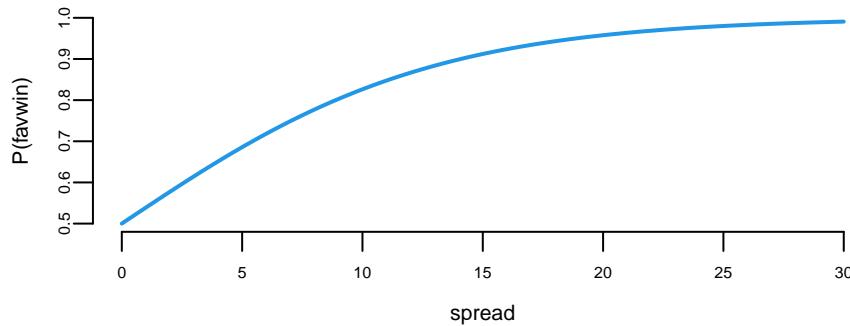
We start by loading the data and visualizing it.

favwin	favscr	undscr	spread	favhome	fregion	uregion
1	72	61	7.0	0	3	4
1	82	74	7.0	1	3	1
1	87	57	17.0	1	3	3
0	69	70	9.0	1	3	3
0	77	79	2.5	0	2	3
1	91	65	9.0	0	3	4



Does the Vegas point spread predict whether the favorite wins or not? The histogram shows the distribution of point spreads for games where the favorite won (turquoise) versus games where the favorite lost (purple). The boxplot provides another view of this relationship. Let's fit a logistic regression model to quantify this relationship:

term	estimate	std.error	statistic	p.value
spread	0.16	0.01	11	0



The β measures how our log-odds change. For this model, we have $\beta = 0.156$, meaning that for every one point increase in the point spread, the log-odds of the favorite winning increases by 0.156.

Now, we can use the model to predict the probability of the favorite winning for a new game with a point spread of 8 or 4.

```
predict(nbareg, newdata = data.frame(spread = c(8, 4)), type =
  "response")
##      1     2
## 0.78 0.65
```

The code above simply “Plugs-in” the values for the new game into our logistic regression

$$P(\text{favwin} \mid \text{spread}) = \frac{e^{\beta x}}{1 + e^{\beta x}}$$

We can calculate it manually as well.

```
exp(0.156 * 8) / (1 + exp(0.156 * 8))
## 0.78
exp(0.156 * 4) / (1 + exp(0.156 * 4))
## 0.65
```

Check that when $\beta = 0$ we have $p = \frac{1}{2}$.

Given our new values spread= 8 or spread= 4, the win probabilities are 78% and 65%, respectively. Clearly, the bigger spread means a higher chance of winning.

Notice that the predict function returns a numeric value between 0 and 1. However, if we want to make a decision (to bet or not to bet), we need to have

a binary outcome. A simple method to move between the predicted probability and binary value is to use thresholding.

$$\hat{y}_i = \begin{cases} 1 & \text{if } \hat{p}_i > \alpha \\ 0 & \text{if } \hat{p}_i \leq \alpha \end{cases}$$

where α is a threshold value. A typical choice is $\alpha = 0.5$.

Now let's calculate the number of correct predictions using threshold $\alpha = 0.5$. R has a convenient `table` function that can summarize the counts of the predicted and actual values in a table.

```
table(NBA$favwin, as.integer(predict(nbareg, type = "response"))
  <- > 0.5), dnn = c("Actual", "Predicted"))
##           Predicted
## Actual      1
##       0 131
##       1 422
```

Our model gets 0.7631103 of the predictions correctly. This number is called *accuracy* of the model.

13.2 Confusion Matrix

We will analyse the tennis data set to show what is the decision boundary for the logistic regression model. The decision boundary is the line that separates the two classes. It is defined as the line where the probability of the favorite winning is 0.5. Then we will use the confusion matrix to evaluate the performance of the model.

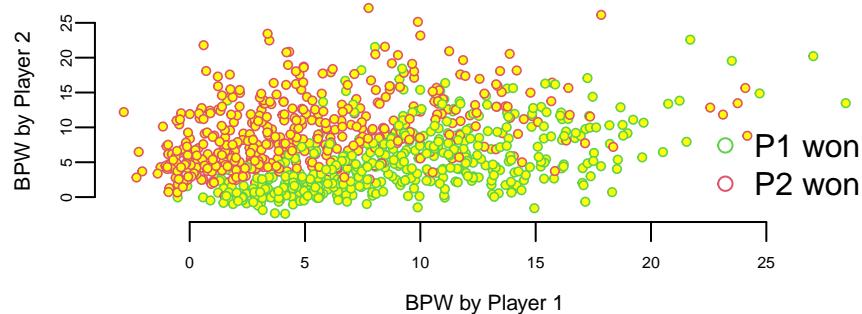
Example 13.2 (Logistic Regression for Tennis Classification). Data science plays a major role in tennis, you can learn about recent AI tools developed by IBM from this [Yahoo! Finance Article](#).

We will analyze the [Tennis Major Tournament Match Statistics Data Set](#) from the UCI ML repository. The data set has one per each game from four major Tennis tournaments in 2013 (Australia Open, French Open, US Open, and Wimbledon).

Let's look at a few columns of the randomly selected five rows of the data

	Player1	Player2	Round	Result	gender	surf
532	Florian Mayer	Juan Monaco	1	1	M	Hard
816	L.Kubot	J.Janowicz	5	0	M	Grass
431	Svetlana Kuznetsova	Ekaterina Makarova	1	1	W	Clay
568	Marcos Baghdatis	Go Soeda	1	1	M	Hard
216	Mandy Minella	Anastasia Pavlyuchenkova	2	0	W	Hard

We have data for 943 matches and for each match we have 44 columns, including names of the players, their gender, surface type and match statistics. Let's look at the number of break points won by each player. We will plot BPW (break points won) by each player on the scatter plot and will colorize each dot according to the outcome



We can clearly see that the number of break points won is a clear predictor of the match outcome. This is obvious and follows from the rules; to win a match, a player must win break points. Now, we want to understand the impact of winning a break point on the overall match outcome. We do it by building a logistic regression model

```
which(is.na(d$BPW.1)) # there is one row with NA value for the
#<- BPW.1 value and we remove it
## 171
d <- d[-171, ]
n <- dim(d)[1]
m <- glm(Result ~ BPW.1 + BPW.2 - 1, data = d, family =
#<- "binomial")
m %>%
```

```
tidy() %>%
kable()
```

term	estimate	std.error	statistic	p.value
BPW.1	0.40	0.03	15	0
BPW.2	-0.42	0.03	-15	0

The predicted values are stored in the `fitted.values` field of the model object. Those are the probabilities of player 1 winning the match. We need to convert them to binary predictions using 0.5 as a threshold for our classification.

```
table(d$result, as.integer(m$fitted.values > 0.5), dnn =
  c("Actual", "Predicted"))
##           Predicted
## Actual      0   1
##          0 416 61
##          1 65 400
```

This table shows the number of correct and incorrect predictions for each class. The rows are the actual outcomes and the columns are the predicted outcomes. The first row shows the number of matches where player 1 won and the model predicted that player 1 won. The second row shows the number of matches where player 1 lost and the model predicted that player 1 lost. Thus, our model got $(400+416)/942 = 86.6242038\%$ of the predictions correctly! The accuracy is the ratio of the number of correct predictions to the total number of predictions.

This table is called *confusion matrix*. It is a table that shows the number of correct and incorrect predictions for each class. The rows are the actual outcomes and the columns are the predicted outcomes. Formally, it is defined as

Table 13.5: Confusion Matrix. TPR - True Positive Rate, FPR - False Positive Rate, TNR - True Negative Rate, FNR - False Negative Rate.

	Predicted: YES	Predicted: NO
Actual: YES	TPR	FNR
Actual: NO	FPR	TNR

Fundamentally, logistic regression attempts to construct a linear boundary that separates the two classes. In our case, we have two predictors $x_1 = \text{BPW.1}$

and $x_2 = \text{BPW.2}$ and our model is

$$\log\left(\frac{p}{1-p}\right) = \beta_1 x_1 + \beta_2 x_2,$$

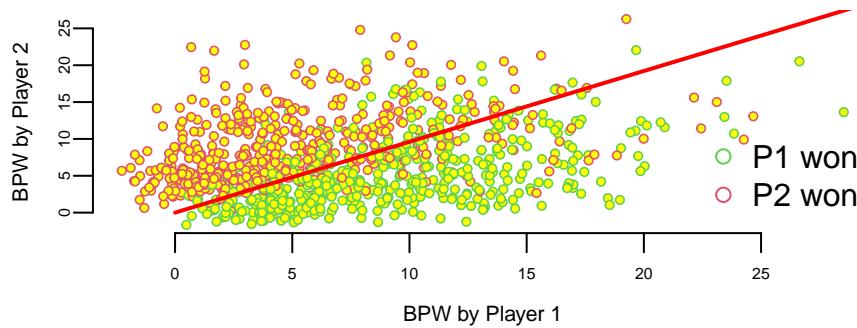
where p is the probability of player 1 winning the match. We want to find the line along which the probability is $1/2$, meaning that $p/(1-p) = 1$ and $\log(p/(1-p)) = 0$, thus the equation for the line is $\beta_1 x_1 + \beta_2 x_2 = 0$ or

$$x_2 = \frac{-\beta_1}{\beta_2} x_1$$

Let's see the line found by the `glm` function

```
plot(d$BPW.1 + rnorm(n), d$BPW.2 + rnorm(n),
  pch = 21, col = d$Result + 2, cex = 0.6, bg = "yellow", lwd =
  0.8,
  xlab = "BPW by Player 1", ylab = "BPW by Player 2"
)

x <- seq(0, 30, length.out = 200)
y <- -m$coefficients[1] * x / m$coefficients[2]
lines(x, y, lwd = 2, col = "red")
```



There are a couple of observations. First, the effect of a break point on the game outcome is significant and symmetric; the effect of losing a break point is the same as the effect of winning one. We also can interpret the effect of winning a break point in the following way. We will keep $\text{BPW.2} = 0$ and will calculate what happens to the probability of winning when BPW.1 changes from 0 to 1. The odds ratio for player 1 winning when $\text{BPW.1} = 0$ is $\exp(0)$ which is 1, meaning that the probability that P1 wins is $1/2$. Now when $\text{BPW.1} = 1$, the odds ratio is

```
exp(0.4019)
## 1.5
```

We can calculate probability of winning from the regression equation

$$\frac{p}{1-p} = 1.5, \quad p = 1.5(1-p), \quad 2.5p = 1.5, \quad p = 0.6$$

Thus probability of winning goes from 50% to 60%, we can use `predict` function to get this result

```
predict.glm(m, newdata = data.frame(BPW.1 = c(0), BPW.2 = c(0)),
            type = "response")
## 1
## 0.5
predict.glm(m, newdata = data.frame(BPW.1 = c(1), BPW.2 = c(0)),
            type = "response")
## 1
## 0.6
```

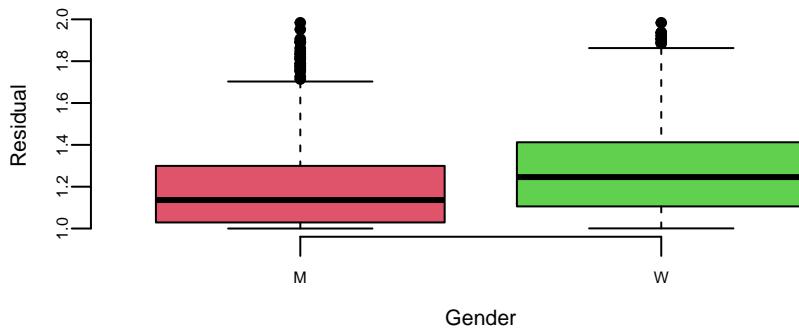
What happens to the chances of winning when P1 wins three more break points compared to the opponent

```
predict.glm(m, newdata = data.frame(BPW.1 = c(0), BPW.2 = c(0)),
            type = "response")
## 1
## 0.5
predict.glm(m, newdata = data.frame(BPW.1 = c(3), BPW.2 = c(0)),
            type = "response")
## 1
## 0.77
```

Chances go up by 27%.

Tennis is arguably the sport in which men and women are treated equally. Both men's and women's matches are shown during prime-time on TV, and they both have the same prize money. However, one of the comments you hear often is that women's matches are "less predictable", meaning that an upset (when the favorite loses) is more likely to happen in a women's match compared to men's matches. We can test this statement by looking at the residuals. The larger the residual the less accurate our prediction was.

```
outlind <- which(d$res < 2)
boxplot(d$res[outlind] ~ d$gender[outlind], col = c(2, 3), xlab
       = "Gender", ylab = "Residual")
```



Let's do a formal t-test on the residuals for men's and women's matches

```
men <- d %>%
  filter(res < 2, gender == "M") %>%
  pull(res)
women <- d %>%
  filter(res < 2, gender == "W") %>%
  pull(res)
t.test(men, women, alternative = "two.sided")
##
## Welch Two Sample t-test
##
## data: men and women
## t = -5, df = 811, p-value = 0.000003
## alternative hypothesis: true difference in means is not equal
## to 0
## 95 percent confidence interval:
## -0.105 -0.043
## sample estimates:
## mean of x mean of y
##      1.2      1.3
```

The difference of 0.07 between men and women and the statistic value of -4.7 means that the crowd wisdom that women's matches are less predictable is correct. The difference is statistically significant!

13.3 ROC Curve and Confounding Variables

Using default data set, we will illustrate the concept of ROC curve and confounding variables.

Example 13.3 (Horse Race Betting). Horse race betting provides a rich application of logistic regression, where predicting whether the favorite wins illustrates key classification concepts. This example uses data from the Hong Kong Jockey Club, one of the world's largest horse racing operations (see the Benter case study for comprehensive analysis).

The betting public's implied probabilities, derived from pari-mutuel odds, exhibit the well-documented *favorite-longshot bias*: favorites tend to win more often than the odds suggest, while longshots win less often. We can build a logistic regression model to predict whether the favorite wins based on race characteristics.

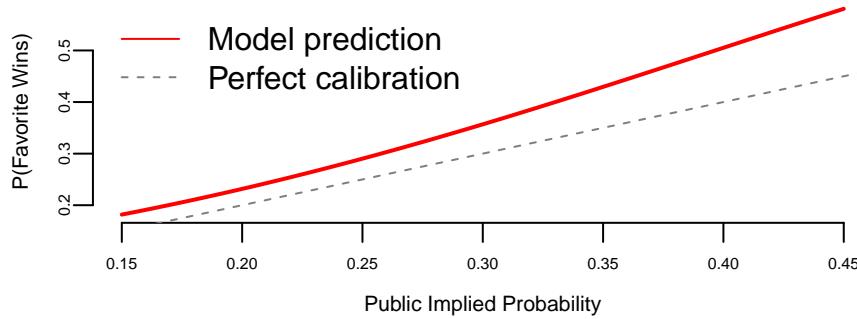
First, we load and prepare the data. We identify the favorite in each race as the horse with the highest implied probability from the betting odds.

```
## Dataset: 5885 races from 1997-11-30 to 2005-08-28
## Favorite win rate: 29.9%
```

Our outcome variable is whether the favorite won (`win = 1`) or not (`win = 0`). The favorite's implied probability from public odds should be a strong predictor; if the market is perfectly calibrated, this probability would equal the true win rate.

term	estimate	std.error	statistic	p.value
(Intercept)	-2.4	0.11	-22	0
public_prob	6.1	0.41	15	0

The positive coefficient on `public_prob` confirms that higher implied probability predicts higher actual win probability. We can visualize this relationship:

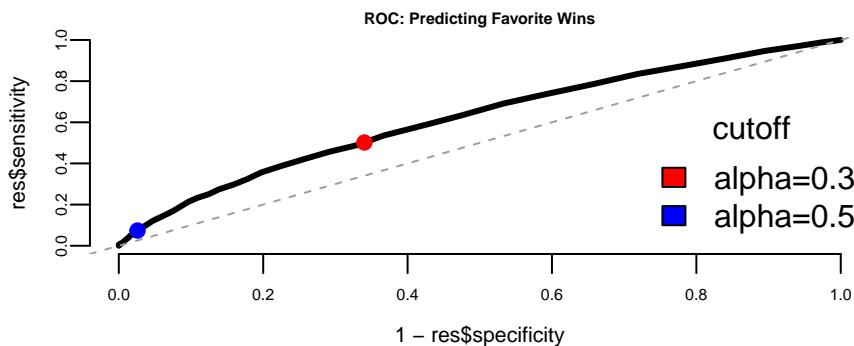


The model prediction lies above the 45-degree line, confirming the favorite-longshot bias: favorites win more often than their odds imply.

Now we build the confusion matrix. In betting, the threshold choice has financial implications: being too aggressive (low threshold) means betting on too many “favorites” that lose, while being too conservative means missing profitable opportunities.

	0	1
0	0.66	0.34
1	0.50	0.50

We use a threshold of 0.3 because the baseline win rate for favorites is around 30%. The ROC curve helps us understand the trade-off between correctly identifying winners (sensitivity) and avoiding false predictions on losers (specificity).



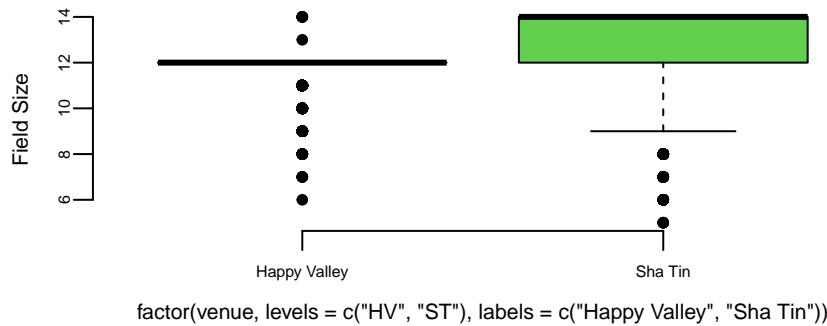
The ROC curve shows reasonable predictive power above the diagonal. The choice of threshold depends on the betting strategy: a lower threshold captures more true winners but also more false positives.

Now let's examine multiple predictors. Hong Kong has two racetracks: Happy Valley (a tight urban track) and Sha Tin (a modern facility with longer straights). We might expect venue to affect favorite performance.

term	estimate	std.error	statistic	p.value
(Intercept)	-2.95	0.32	-9.37	0.00
public_prob	6.32	0.43	14.55	0.00
field_size	0.04	0.02	1.83	0.07
venue_ST	-0.07	0.07	-0.96	0.34

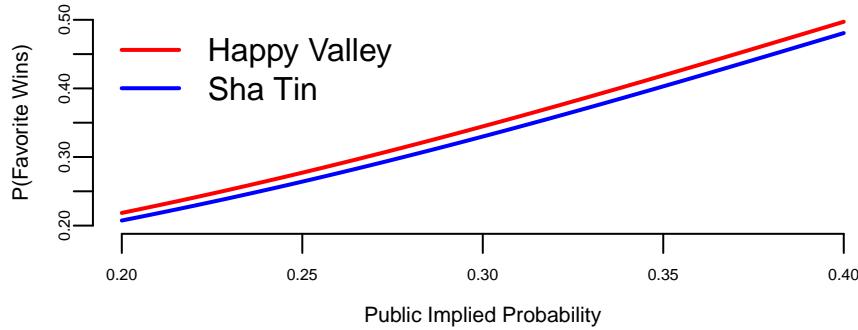
The coefficient for `field_size` is negative, which seems intuitive; in larger fields, there's more competition, so favorites may have harder time winning. But the coefficient for `venue_ST` (Sha Tin) is also negative. Does this mean favorites perform worse at Sha Tin?

Let's check for confounding by examining field sizes at each venue:



Sha Tin races have systematically larger fields. This creates confounding: the apparent venue effect may actually be a field size effect. When we control for field size in the model, the venue effect represents the *residual* impact of venue beyond what's explained by field size.

To see this more clearly, let's compare predictions across venues for races with the same field size:



After controlling for field size, the venue difference is smaller. The remaining gap may reflect track characteristics: Happy Valley's tighter turns could favor front-running favorites who avoid traffic problems.

To summarize: ROC curves help visualize the trade-off between catching winners and avoiding false positives across different probability thresholds. In betting applications, this trade-off has direct financial consequences, making threshold selection a critical business decision. Confounding is also evident: Sha Tin races appear to disadvantage favorites, but this partly reflects larger field sizes rather than track characteristics. Including confounders in the model isolates the true venue effect. These principles from horse racing apply broadly to any classification problem where features are correlated.

Now, a natural question is how to choose the cut-off value α ? In betting, we place a bet when $p(\text{win}) > \alpha$. Here α is our confidence threshold. If we choose $\alpha = 0$ and bet on everything, we'll lose money on bad bets. If we choose $\alpha = 1$, we never bet and make no money. In order to choose an appropriate α , we need to know the payoffs.

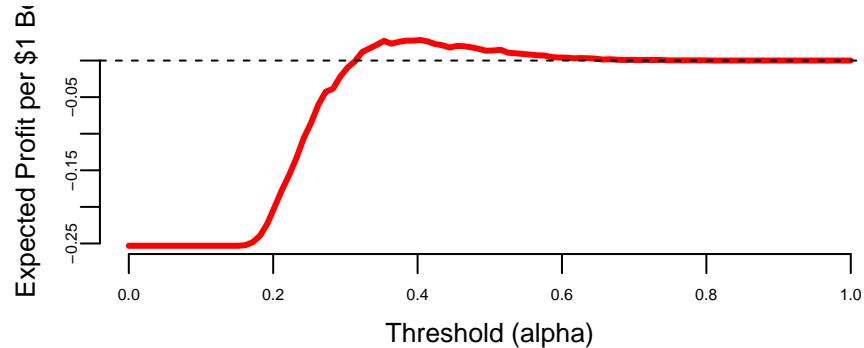
For betting on favorites at typical odds (say, 2.5:1 decimal odds), the pay-off matrix looks like:

Table 13.9: Pay-off matrix for betting

	Win	Lose
Bet	1.5	-1
Don't bet	0	0

If we bet \$1 and win at 2.5:1 odds, we gain \$1.50 profit. If we lose, we lose our \$1 stake. If we don't bet, we neither gain nor lose.

Given this pay-off matrix, we can calculate the expected profit across different thresholds using our horse racing data.



To identify the most effective strategy, we evaluate the expected profit across all possible thresholds α . The expected profit per dollar wagered is calculated as a weighted average of the gains from true positives and the losses from false positives:

$$E[\text{Profit} | \alpha] = P(\text{win}) \cdot \text{Sensitivity}(\alpha) \cdot 1.5 - P(\text{lose}) \cdot (1 - \text{Specificity}(\alpha)) \cdot 1.0$$

where $P(\text{win})$ is the base rate of winners in the training data. This calculation allows us to identify the *profitable threshold range* (where expected profit is positive), the *optimal threshold* (which maximizes profit), and the resulting *maximum expected profit*.

```
## Profitable threshold range: 0.32 to 0.80
## Optimal threshold: 0.40
## Maximum expected profit: $0.028 per $1 wagered
```

This analysis shows that the choice of threshold has direct financial consequences. Unlike academic classification problems where we might default to $\alpha = 0.5$, real-world applications require understanding the asymmetric costs of different types of errors.

Example 13.4 (LinkedIn Study). How to Become an Executive(Irwin 2016; Gan and Fritzler 2016)?

Logistic regression was used to analyze the career paths of about 459,000 LinkedIn members who worked at a [top 10 consultancy](#) between 1990 and 2010 and became a VP, CXO, or partner at a company with at least 200 employees. About 64,000 members reached this milestone, $\hat{p} = 0.1394$, conditional on making it into the database. The goals of the analysis were the following

1. Look at their profiles – educational background, gender, work experience, and career transitions.

2. Build a predictive model of the probability of becoming an executive
3. Provide a tool for analysis of “what if” scenarios. For example, if you are to get a master’s degree, how your jobs perspectives change because of that.

Let’s build a logistic regression model with 8 key features (a.k.a. covariates):

$$\log \left(\frac{p}{1-p} \right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_8 x_8$$

Here p is the probability of “success” – meaning the person reaches VP/CXO/Partner seniority at a company with at least 200 employees. The features to predict the “success” probability are $x_i (i = 1, 2, \dots, 8)$

- x_1 : Metro region: whether a member has worked in one of the top 10 largest cities in the U.S. or globally.
- x_2 : Gender: Inferred from member names: ‘male’, or ‘female’
- x_3 : Graduate education type: whether a member has an MBA from a top U.S. program / a non-top program / a top non-U.S. program / another advanced degree
- x_4 : Undergraduate education type: whether a member has attended a school from the U.S. News national university rankings / a top 10 liberal arts college / a top 10 non-U.S. school
- x_5 : Company count: # different companies in which a member has worked
- x_6 : Function count: # different job functions in which a member has worked
- x_7 : Industry sector count: # different industries in which a member has worked
- x_8 : Years of experience: # years of work experience, including years in consulting, for a member.

The following estimated $\hat{\beta}$ s of features were obtained. With a sample size of 456,000 they are measured rather accurately. Recall that in logistic regression, the coefficients represent the change in log-odds for a unit change in the feature.

Category	Feature	Coefficient ($\hat{\beta}$)
Location	Metro region	0.28
Personal	Gender (Male)	0.31
Education	Graduate education type	1.16
Education	Undergraduate education type	0.22

Category	Feature	Coefficient ($\hat{\beta}$)
Work Experience	Company count	0.14
Work Experience	Function count	0.26
Work Experience	Industry sector count	-0.22
Work Experience	Years of experience	0.09

Here are three main findings

1. Working across job functions, like marketing or finance, is good. Each additional job function provides a boost that, on average, is equal to three years of work experience. Switching industries has a slight negative impact. Learning curve? lost relationships?
2. MBAs are worth the investment. But pedigree matters. *Top five program equivalent to 13 years of work experience!!!*
3. Location matters. For example, NYC helps.

We can also personalize the prediction for predict future possible future executives. For example, Person A (p=6%): Male in Tulsa, Oklahoma, Undergraduate degree, 1 job function for 3 companies in 3 industries, 15-year experience.

Person B (p=15%): Male in London, Undergraduate degree from top international school, Non-MBA Master, 2 different job functions for 2 companies in 2 industries, 15-year experience.

Person C (p=63%): Female in New York City, Top undergraduate program, Top MBA program, 4 different job functions for 4 companies in 1 industry, 15-year experience.

Let's re-design Person B.

Person B (p=15%): Male in London, Undergraduate degree from top international school, Non-MBA Master, 2 different job functions for 2 companies in 2 industries, 15-year experience.

1. Work in one industry rather than two. Increase 3%
2. Undergrad from top 10 US program rather than top international school. 3%
3. Worked for 4 companies rather than 2. Another 4%
4. Move from London to NYC. 4%
5. Four job functions rather than two. 8%. A 1.5x effect.
6. Worked for 10 more years. 15%. A 2X effect.

Choices and Impact (Person B) are shown below

Choices and Impact (Person B) are shown below in Figure 13.2. The chart illustrates the marginal impact of each individual strategic choice (e.g., getting an MBA, moving to NYC) on the probability of becoming an executive,

compared to Person B's baseline of 15%. When all these positive choices are combined, the probability skyrockets to 81%.

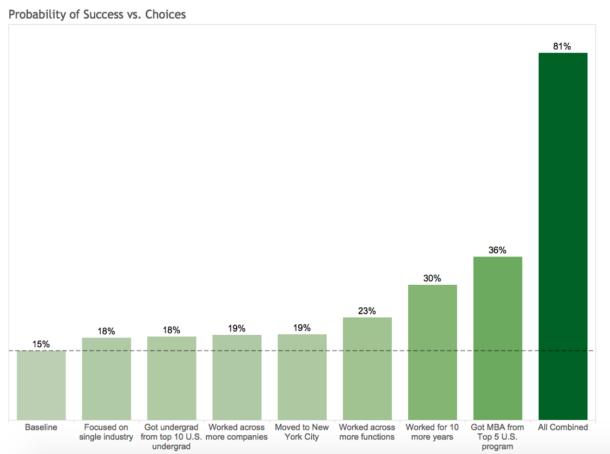


Figure 13.2: Choices and Impact (Person B)

13.4 Imbalanced Data

Often, you have much more observations with a specific label, such a sample is called imbalanced. This is a common problem in real-world classification tasks where one class significantly outnumbers the other(s). For example, in fraud detection, legitimate transactions vastly outnumber fraudulent ones; in medical diagnosis, healthy patients often outnumber those with rare diseases; and in manufacturing, defective products are typically much rarer than non-defective ones.

When dealing with imbalanced data, you should avoid using accuracy as a metric to choose a model. Consider a binary classification problem with 95% of samples labeled as class 1. A naive classifier that simply assigns label 1 to every input will achieve 95% accuracy, making it appear deceptively good while being completely useless for practical purposes.

Instead, more appropriate evaluation metrics should be used. The Receiver Operating Characteristic (ROC) curve plots the true positive rate (sensitivity) against the false positive rate (1-specificity) at various classification thresholds. The Area Under the Curve (AUC) provides a single scalar value that measures the model's ability to distinguish between classes, regardless of the chosen threshold. An AUC of 0.5 indicates random guessing, while 1.0 represents

perfect classification.

The F1 score combines precision and recall into a single score, providing a balanced measure that penalizes models that are either too conservative or too aggressive:

$$F1 = 2 \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

where precision measures the proportion of true positives among predicted positives, and recall measures the proportion of true positives that were correctly identified.

The precision-recall curve is particularly useful for imbalanced datasets, as it plots precision against recall at various thresholds, focusing on the performance of the positive class. Cohen's Kappa measures agreement between predicted and actual classifications while accounting for agreement by chance, making it more robust to class imbalance than accuracy.

To address imbalanced data, several strategies can be employed. Data-level approaches include oversampling, where you synthetically generate more samples of the minority class using techniques like bootstrap sampling with replacement, SMOTE (Synthetic Minority Over-sampling Technique) which creates synthetic examples by interpolating between existing minority class samples, or generative models like GANs or variational autoencoders to create realistic synthetic data. Undersampling reduces the majority class samples, which is particularly effective when the dataset is large enough. Hybrid approaches combine both oversampling and undersampling techniques.

Algorithm-level approaches include cost-sensitive learning, where you assign different misclassification costs to different classes, ensemble methods using techniques like bagging or boosting that can naturally handle imbalanced data, and threshold adjustment to modify the classification threshold to optimize for specific metrics like F1-score.

The choice of approach depends on the specific problem, available data, and computational resources. It is often beneficial to experiment with multiple techniques and evaluate their performance using appropriate metrics rather than relying solely on accuracy.

13.5 Kernel Trick

While logistic regression is a powerful linear classifier, there are cases where the classes are not linearly separable in the original feature space. For example, one class might be encircled by another. In such situations, we need to extend our linear methods to handle non-linear decision boundaries.

The kernel trick is a method of using a linear classifier to solve a non-linear problem. The idea is to map the data into a higher dimensional space, where it becomes linearly separable. The kernel trick is to use a kernel function $K(x_i, x_j)$ to calculate the inner product of two vectors in the higher dimensional space without explicitly calculating the mapping $\phi(x_i)$ and $\phi(x_j)$. The kernel function is defined as $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. The most popular kernel functions are polynomial kernel $K(x_i, x_j) = (x_i^T x_j)^d$ and Gaussian kernel $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$. The kernel trick is used in Support Vector Machines (SVM) and Gaussian Processes (GP).

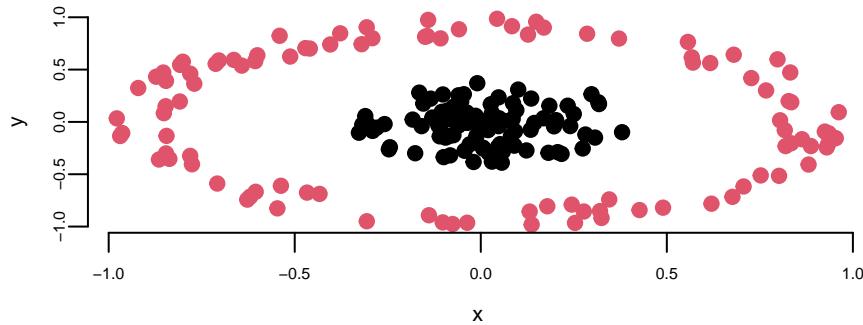


Figure 13.3

The data in Figure 13.3 is not linearly separable in two dimensions; however, projecting it into a three-dimensional space via the following transformation allows for a linear separation:

$$\begin{aligned}\phi : R^2 &\longrightarrow R^3 \\ (x_1, x_2) &\longmapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2),\end{aligned}$$

and attempt to linearly separate the transformed data, the decision boundaries become hyperplanes in R^3 , expressed as $\omega^T z + b = 0$. In terms of the original variables x , these boundaries take the form:

$$\omega_1 x_1^2 + \omega_2 \sqrt{2}x_1x_2 + \omega_3 x_2^2 = 0,$$

which corresponds to the equation of an ellipse. This demonstrates that we can apply a linear algorithm to transformed data to achieve a non-linear decision boundary with minimal effort.

Now, consider what the algorithm is actually doing. It relies solely on the Gram matrix K of the data. Once K is computed, the original data can be

discarded:

$$K = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & \cdots \\ x_2^T x_1 & \ddots & \\ \vdots & & \end{bmatrix}_{n \times n} = X X^T,$$

where $X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}_{n \times d}$.

Here, X , which contains all the data, is referred to as the design matrix.

When we map the data using ϕ , the Gram matrix becomes:

$$K = \begin{bmatrix} \phi(x_1)^T \phi(x_1) & \phi(x_1)^T \phi(x_2) & \cdots \\ \phi(x_2)^T \phi(x_1) & \ddots & \\ \vdots & & \end{bmatrix}.$$

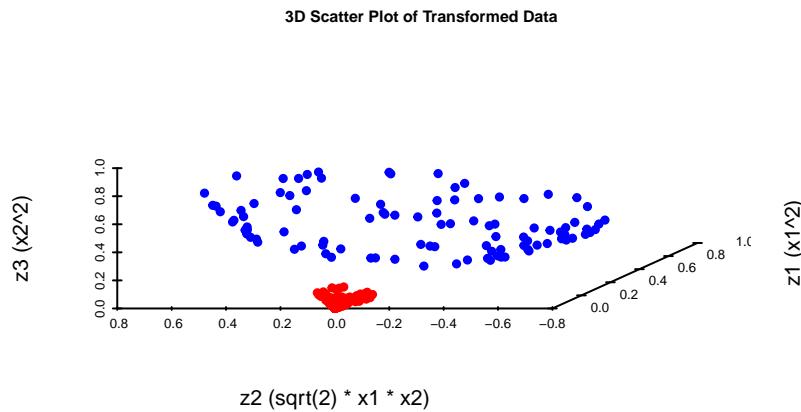
Let us compute these inner products explicitly. For vectors r and s in R^3 corresponding to a and b , respectively:

$$\begin{aligned} \langle r, s \rangle &= r_1 s_1 + r_2 s_2 + r_3 s_3 \\ &= a_1^2 b_1^2 + 2a_1 a_2 b_1 b_2 + a_2^2 b_2^2 \\ &= \langle a, b \rangle^2. \end{aligned}$$

Thus, instead of explicitly mapping the data via ϕ and then computing the inner product, we can compute it directly in one step, leaving the mapping ϕ implicit. In fact, we do not even need to know ϕ explicitly; all we require is the ability to compute the modified inner product. This modified inner product is called a kernel, denoted $K(x, y)$. The matrix K , which contains the kernel values for all pairs of data points, is also referred to as the kernel matrix.

Since the kernel itself is the primary object of interest, rather than the mapping ϕ , we aim to characterize kernels without explicitly relying on ϕ . Mercer's Theorem provides the theoretical guarantee for this: it states that for any symmetric, positive-definite function $K(x, y)$, there exists a mapping ϕ such that $K(x, y) = \langle \phi(x), \phi(y) \rangle$. This ensures we can implicitly work in a high-dimensional space just by defining a valid kernel function.

Let's implement it



13.6 Generalized linear models

Logistic regression is one member of a larger family of models called generalized linear models (GLMs). A GLM has three components:

1. a distributional assumption for Y from the exponential family (e.g., Normal, Bernoulli, Poisson, Gamma),
2. a linear predictor $\eta_i = \mathbf{x}_i^\top \boldsymbol{\beta}$,
3. a link function $g(\cdot)$ connecting the mean response to the linear predictor, $g(\mathbb{E}(Y_i | \mathbf{x}_i)) = \eta_i$.

This framework makes it easy to move between problem types (continuous, binary, counts, positive-valued responses) while keeping a common estimation story based on likelihood and its negative log (Chapter 11).

The link function encodes how predictors move the mean response. Common choices include:

Link Function	Distribution	Use Case
logit	Bernoulli/binomial	logistic regression for binary outcomes
probit	Bernoulli/binomial	Normal-CDF alternative for binary data
log	Poisson, Gamma	count data and positive-valued responses
inverse	Gamma	positive continuous data (alternative parameterization)

For count outcomes $Y_i \in \{0, 1, 2, \dots\}$, a standard GLM is *Poisson regression*:

$$Y_i | x_i \sim \text{Poisson}(\lambda_i), \quad \log \lambda_i = x_i^\top \beta.$$

The log link ensures $\lambda_i > 0$. Coefficients have a multiplicative interpretation: increasing a covariate by one unit changes the mean count by a factor of $\exp(\beta_j)$, holding other predictors fixed.

For continuous responses that are strictly positive and right-skewed (e.g., waiting times, costs), a common GLM choice is *Gamma regression*. One parameterization is

$$Y_i | x_i \sim \text{Gamma}(\text{shape}, \text{rate}), \quad E(Y_i | x_i) = \mu_i, \quad \log \mu_i = x_i^\top \beta,$$

where the log link again enforces positivity of the mean. Alternatives include an inverse link, depending on the modeling context.

13.6.1 Deviance and model comparison

In GLMs, goodness-of-fit is often summarized by the deviance, a likelihood-based measure that quantifies how well a fitted model explains the data relative to a benchmark. Understanding deviance requires first understanding what we mean by a saturated model.

A *saturated model* is one that fits the data perfectly by using as many parameters as there are observations. For binary logistic regression with n distinct covariate patterns, a saturated model assigns a separate probability parameter \hat{p}_i to each observation, setting $\hat{p}_i = y_i$. This achieves the maximum possible log-likelihood for the given data structure, though at the cost of using n parameters and having no degrees of freedom left for testing or prediction. The

saturated model serves as an upper bound on how well any model can fit the observed data.

Formally, the *deviance* of a fitted GLM is defined as

$$D = 2 \left[\ell(\text{saturated}) - \ell(\hat{\beta}) \right],$$

where $\ell(\text{saturated})$ is the log-likelihood of the saturated model and $\ell(\hat{\beta})$ is the log-likelihood of the fitted model with parameter estimates $\hat{\beta}$. The factor of 2 appears for historical reasons related to asymptotic chi-squared distributions and makes the deviance directly comparable to likelihood ratio test statistics.

For binary logistic regression, this becomes

$$D = 2 \sum_{i=1}^n \left[y_i \log \left(\frac{y_i}{\hat{p}_i} \right) + (1 - y_i) \log \left(\frac{1 - y_i}{1 - \hat{p}_i} \right) \right],$$

where $\hat{p}_i = \sigma(x_i^\top \hat{\beta})$ are the fitted probabilities. When $y_i \in \{0, 1\}$, terms with $y_i \log(y_i)$ or $(1 - y_i) \log(1 - y_i)$ equal zero by convention ($0 \log 0 = 0$), so each observation contributes either $-2 \log(\hat{p}_i)$ when $y_i = 1$ or $-2 \log(1 - \hat{p}_i)$ when $y_i = 0$.

Lower deviance indicates better fit: a deviance of zero means the model fits as well as the saturated model. In practice, deviance is most useful for comparing nested models rather than as an absolute measure. Suppose we have two nested models with $M_1 \subset M_2$, where model M_1 has p_1 parameters and model M_2 has $p_2 > p_1$ parameters. The difference in deviances

$$\Delta D = D_1 - D_2 = 2 \left[\ell(\hat{\beta}_2) - \ell(\hat{\beta}_1) \right]$$

follows approximately a $\chi^2_{p_2 - p_1}$ distribution under the null hypothesis that the simpler model M_1 is adequate. This provides a formal likelihood ratio test: if ΔD exceeds the critical value from the chi-squared distribution, we reject M_1 in favor of the more complex M_2 .

This framework generalizes beyond logistic regression to all GLMs. For Poisson regression with counts y_i and fitted means $\hat{\mu}_i$, the deviance takes the form

$$D = 2 \sum_{i=1}^n \left[y_i \log \left(\frac{y_i}{\hat{\mu}_i} \right) - (y_i - \hat{\mu}_i) \right].$$

The specific formula changes with the exponential family distribution, but the interpretation remains consistent: deviance measures twice the log-likelihood gap between the fitted and saturated models, enabling principled comparison of nested specifications. This connects naturally to the model selection criteria in Chapter 16, where penalized versions of deviance like AIC and BIC trade off fit against model complexity.

13.7 Bayesian Logistic Regression with The Polya-Gamma Distribution

To perform Bayesian inference for logistic regression, we often need to sample from the posterior distribution of the coefficients β . Conceptually this follows the same updating pattern as in Chapter 3: posterior \propto likelihood \times prior. It also connects to the likelihood-to-loss framing in Chapter 11: the same likelihood that defines the posterior also defines an optimization objective through its negative log. However, the logistic likelihood does not have a conjugate prior, making direct sampling difficult. A powerful modern approach uses data augmentation with Polya-Gamma variables to make the sampling exact and efficient.

The Polya-Gamma distribution, denoted as PG(b,c), is carefully constructed as a subset of infinite convolutions of gamma distributions (essentially, a weighted sum of an infinite number of independent Gamma variables)(Nicholas G. Polson, Scott, and Windle 2013). A random variable X follows a Polya-Gamma distribution with parameters $b > 0$ and $c \in \mathbb{R}$ if:

$$X \stackrel{d}{=} \frac{1}{2\pi^2} \sum_{k=1}^{\infty} \frac{g_k}{(k - 1/2)^2 + c^2/(4\pi^2)},$$

where $g_k \sim \text{Ga}(b, 1)$ are independent gamma random variables, and $\stackrel{d}{=}$ indicates equality in distribution.

The Polya-Gamma family exhibits several remarkable properties that make it ideal for data augmentation:

1. **Laplace Transform:** For $\omega \sim \text{PG}(b, 0)$, the Laplace transform is $E(\exp(-\omega t)) = \cosh^{-b}(\sqrt{t}/2)$.
2. **Exponential Tilting:** The general PG(b,c) distribution arises through exponential tilting of the PG(b,0) density:

$$p(x|b, c) = \frac{\exp(-c^2x/2)p(x|b, 0)}{E(\exp(-c^2\omega/2))},$$

where the expectation is taken with respect to PG(b,0)

3. **Convolution Property:** The family is closed under convolution for random variates with the same tilting parameter
4. **Known Moments:** All finite moments are available in closed form, with the expectation given by:

$$E(\omega) = \frac{b}{2c} \tanh(c/2) = \frac{b}{2c} \frac{e^c - 1}{1 + e^c}.$$

Computational Advantage

The known moments and convolution properties make the Polya-Gamma distribution computationally tractable and theoretically well-behaved.

13.7.1 The Data-Augmentation Strategy

The core of the Polya-Gamma methodology rests on a fundamental integral identity that represents binomial likelihoods as mixtures of Gaussians(Nicholas G. Polson, Scott, and Windle 2013). The key theorem states:

Theorem 1: For $b > 0$ and $a \in \mathbb{R}$, the following integral identity holds:

$$\frac{(e^\psi)^a}{(1 + e^\psi)^b} = 2^{-b} e^{\kappa\psi} \int_0^\infty e^{-\omega\psi^2/2} p(\omega) d\omega$$

where $\kappa = a - b/2$, and $p(\omega)$ is the density of $\omega \sim PG(b, 0)$ (Nicholas G. Polson, Scott, and Windle 2013).

Moreover, the conditional distribution $p(\omega|\psi)$ is also in the Polya-Gamma class: $(\omega|\psi) \sim PG(b, \psi)$ (Nicholas G. Polson, Scott, and Windle 2013).

This integral identity leads directly to a simple two-step Gibbs sampler for Bayesian logistic regression(Nicholas G. Polson, Scott, and Windle 2013). For a dataset with observations $y_i | \psi_i \sim \text{Binom}(n_i, 1/(1+e^{-\psi_i}))$ where $\psi_i = x_i^T \beta$, and a Gaussian prior $\beta \sim N(b, B)$, the algorithm iterates:

1. **Sample auxiliary variables:** $(\omega_i|\beta) \sim PG(n_i, x_i^T \beta)$ for each observation
2. **Sample parameters:** $(\beta|y, \omega) \sim N(m_\omega, V_\omega)$ where:
 - $V_\omega = (X^T \Omega X + B^{-1})^{-1}$
 - $m_\omega = V_\omega (X^T \kappa + B^{-1} b)$
 - $\kappa = (y_1 - n_1/2, \dots, y_n - n_n/2)$
 - $\Omega = \text{diag}(\omega_1, \dots, \omega_n)$

This approach requires only Gaussian draws for the main parameters and Polya-Gamma draws for a single layer of latent variables, making it significantly simpler than previous methods(Nicholas G. Polson, Scott, and Windle 2013).

We can visualize the hierarchical structure of this data augmentation strategy as follows:

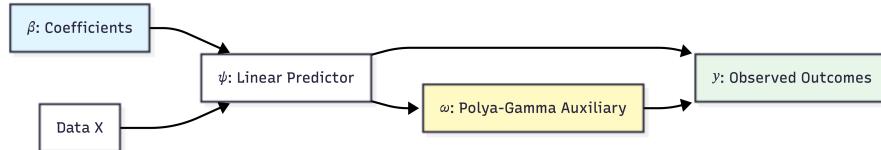


Figure 13.4: Hierarchical structure of Polya-Gamma Data Augmentation

The practical success of the Polya-Gamma method depends on efficient simulation of Polya-Gamma random variables(Nicholas G. Polson, Scott, and Windle 2013). The authors developed a sophisticated accept-reject sampler based on the alternating-series method of Devroye (Devroye 1986). For the fundamental PG(1,c) case, the sampler uses exponential and inverse-Gaussian draws as proposals, achieving an acceptance probability uniformly bounded below at 0.99919. This high acceptance rate requires no tuning for optimal performance. The acceptance criterion is evaluated using iterative partial sums, making the algorithm both efficient and robust.

For integer values of b , $PG(b,z)$ random variables are generated by summing b independent $PG(1,z)$ draws, exploiting the convolution property. This approach maintains efficiency for moderate values of b , though computational cost scales linearly with the total number of counts in negative binomial applications.

The `BayesLogit` package provides efficient tools for sampling from the Polya-Gamma distribution. The current version (2.1) focuses on core functionality: sampling from the Polya-Gamma distribution through the `rpg()` function and its variants.

We demonstrate the `BayesLogit` using a simulated data example. We generate $n = 100$ observations with two predictors (plus an intercept term). The predictor matrix X consists of an intercept column and two columns of independent standard normal random variables. The true regression coefficients are set to $\beta = (-0.5, 1.2, -0.8)$. The binary outcomes y_i are generated from a binomial distribution where the success probability for each observation follows the logistic transformation $p_i = 1/(1 + \exp(-x_i^T \beta))$. This setup allows us to verify that the Bayesian inference procedure can recover the true parameter values from the simulated data.

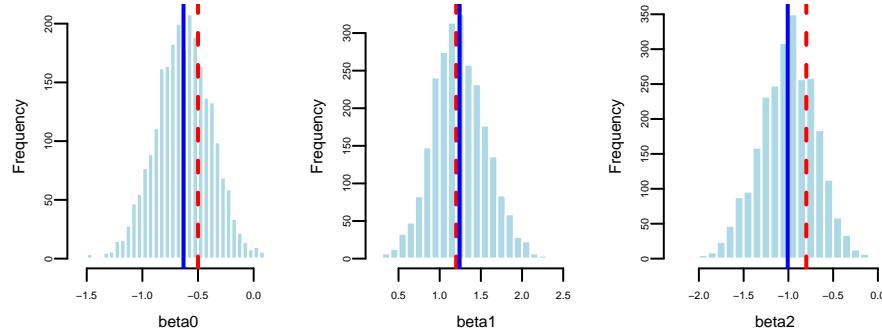


Figure 13.5: Posterior distributions for Bayesian Logistic Regression with Polya-Gamma Data Augmentation. Red line shows true value, blue line shows posterior mean.

The package offers several sampling methods for ω .

- `rpg()`: Main function that automatically selects the best method
- `rpg.devroye()`: Devroye-like method for integer h values
- `rpg.gamma()`: Sum of gammas method (slower but works for all parameters)
- `rpg.sp()`: Saddlepoint approximation method

In the example above we use the automatic function `rpg`.

13.8 Bayesian Analysis of Horse Race Betting

We now apply the Polya-Gamma methodology to the horse racing data from Example 13.3, where we predict whether the race favorite wins. The Bayesian approach provides posterior distributions for the coefficients, quantifying uncertainty in our predictions, which is essential when making betting decisions.

The implementation uses Polya-Gamma augmentation to efficiently sample from the posterior distribution. The full R code for this analysis is available in `case_studies/betting/benter.R`, which implements the Gibbs sampler and generates all results shown below.

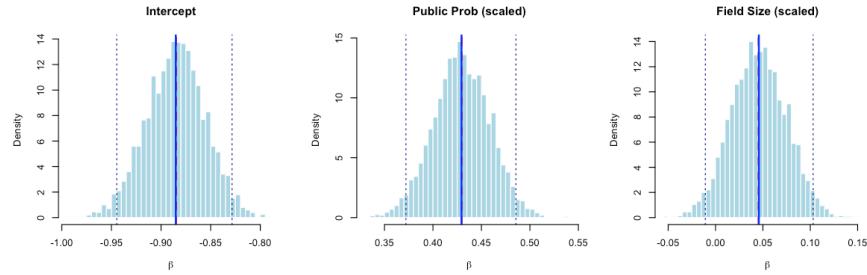


Figure 13.6: Posterior distributions for horse racing logistic regression coefficients. Red dashed lines show MLE estimates; the Bayesian posteriors capture uncertainty around these point estimates.

The posterior distributions (Figure 13.6) reveal important insights for betting strategy. Table 13.12 shows the posterior summaries compared to maximum likelihood estimates:

Table 13.12: Bayesian posterior summaries for horse racing logistic regression

Parameter	Posterior Mean	Posterior SD	95% CI Lower	95% CI Upper	MLE
Intercept	-0.885	0.030	-0.945	-0.828	-0.885
Public Prob (scaled)	0.429	0.029	0.372	0.486	0.430
Field Size (scaled)	0.046	0.029	-0.011	0.103	0.045

The Bayesian analysis confirms our findings: the public probability is a strong positive predictor of favorite success (posterior mean = 0.429, 95% CI: [0.372, 0.486]). The field size effect is weakly positive but uncertain, with the 95% credible interval spanning from -0.011 to 0.103, and only a 5.4% posterior probability that this effect is negative. This uncertainty is valuable for betting applications: we can propagate the posterior distribution through the Kelly criterion to obtain a *distribution* of optimal bet sizes rather than a single point estimate, leading to more robust position sizing.

13.8.1 Theoretical Guarantees

The Polya-Gamma Gibbs sampler enjoys strong theoretical properties that distinguish it from many other MCMC approaches (Nicholas G. Polson, Scott,

and Windle 2013). Perhaps most importantly, Choi and Hobert (2013) proved that the sampler is *uniformly ergodic*, a property that guarantees geometric convergence to the stationary distribution regardless of the starting point. This uniform ergodicity result ensures that central limit theorems hold for Monte Carlo averages computed from the sampler output, enabling valid inference and standard error calculations for posterior quantities of interest.

The property of requiring no tuning parameters represents a significant practical advantage. Traditional Metropolis-Hastings algorithms demand careful calibration of proposal distributions, a task that typically requires pilot runs, domain expertise, and iterative adjustment (Roberts and Rosenthal 2001). Poorly tuned proposals can lead to slow mixing or even apparent convergence to incorrect distributions, and diagnosing such failures often proves difficult. The Polya-Gamma sampler sidesteps these concerns entirely: each conditional distribution in the Gibbs sampling scheme is fully specified by the model, leaving no algorithmic parameters for the practitioner to adjust. This makes the method accessible to users without deep expertise in MCMC diagnostics and eliminates a common source of errors in applied Bayesian analysis.

Equally important is the property of exact sampling from the target posterior distribution. Many popular alternatives to MCMC, such as variational inference or Laplace approximations, sacrifice exactness for computational speed, producing samples from an approximating distribution rather than the true posterior. While such approximations work well in many settings, they can introduce systematic biases that are difficult to quantify, particularly in hierarchical models where the posterior may exhibit complex dependencies. The Polya-Gamma Gibbs sampler, by contrast, produces draws from the exact posterior distribution. Combined with the ergodicity guarantee, this means that averages of the sampled values converge to true posterior expectations as the number of iterations increases, providing asymptotically exact inference without the approximation errors inherent in alternative methods.

Important Note

The theoretical guarantees hold under standard regularity conditions, and the method requires proper prior specification for optimal performance.

The Polya-Gamma methodology extends naturally to negative binomial regression through direct application of the same data-augmentation scheme. For multinomial logistic models, the approach can be extended through partial difference of random utility models(Windle, Polson, and Scott 2014). The framework also allows seamless incorporation of random effects structures in mixed effects models, and provides efficient inference for spatial count data models.

Recent developments have expanded the methodology's applicability. Y.

Zhang, Datta, and Banerjee (2018) applied it to Gaussian process classification that relies on scalable variational approaches using Polya-Gamma augmentation. T. Ye et al. (2023) demonstrated integration with neural network architectures by adding a Gaussian process layer with Pólya-Gamma augmentation to deep neural retrieval models for improved calibration and uncertainty estimation. Frühwirth-Schnatter and Wagner (2010) showed application to dynamic binary time series models through state-space formulations.

The Polya-Gamma methodology represents a fundamental advancement in Bayesian computation for logistic models, combining theoretical elegance with practical efficiency. Its introduction of the Polya-Gamma distribution class and the associated data-augmentation strategy has enabled routine application of Bayesian methods to complex hierarchical models that were previously computationally prohibitive.

As computational demands continue to grow and models become increasingly complex, the Polya-Gamma methodology's advantages become even more pronounced, establishing it as an essential tool in the modern Bayesian statistician's toolkit (Tiao 2019). Ongoing research continues to extend the Polya-Gamma methodology to new domains, including high-dimensional settings, nonparametric models, and integration with modern machine learning frameworks.



14

Tree Models

Imagine you're a jewelry appraiser tasked with determining a diamond's value. You might follow a series of questions: Is the carat weight above 1.0? If yes, is the clarity VS1 or better? Each question leads to another, creating a decision path that eventually arrives at a price estimate. This is precisely how decision trees work—they mirror our natural decision-making process by creating a flowchart of if-then rules.

Logistic regression is a deliberately structured model: it trades flexibility for interpretability and stable estimation. Tree-based methods reverse that trade-off by letting the data determine interactions and nonlinearities. The cost is that algorithmic choices (splitting, pruning, ensembling) become part of the statistical model. This chapter introduces trees as a first major step into nonparametric prediction.

We've used decision trees before to describe the decision-making process as a sequence of actions and conditions. In this section, we'll use decision trees to make predictions. You can think of a prediction as a decision task, where you need to decide which value of y to use for a given x . Similar to a decision tree, a predictive tree model is a nested sequence of if-else statements that map any input data point x to a predicted output y . Each if-else statement checks a feature of x and sends the data left or right along the tree branch. At the end of the branch, a single value of y is predicted.

Figure 14.1 shows a decision tree for predicting a chess piece given a four-dimensional input vector that describes the types of moves available to the piece. The tree is a sequence of nested if-else statements that check the values of the input vector. The tree has six leaves, one for each of the chess pieces and has a depth of four. The tree is a predictive model that maps a four-dimensional input vector to a single output categorical value with six possible values.

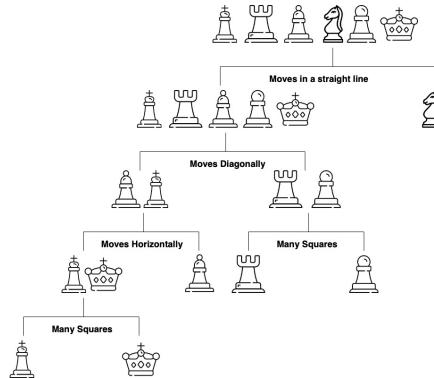


Figure 14.1: Elementary tree scheme; visualization of the splitting process

The prediction mechanism is straightforward: traverse the tree from the root to a leaf node following the conditional logic. The process of building a tree, given a set of training data, is more complicated and has three main components:

1. **Splitting.** The process of dividing the training data into subsets based on the value of a single feature. The goal is to create subsets that are as homogeneous as possible. The subsets are then used to create the nodes of the tree.
2. **Stopping.** The process of deciding when to stop splitting. The goal is to create a tree that is as accurate as possible without overfitting the training data.
3. **Pruning.** The process of removing nodes from the tree that do not improve the accuracy of the tree. The goal is to create a tree that is as accurate as possible without overfitting the training data.

The crux of the tree-building process lies in splitting: determining the optimal feature and threshold that best separates the data. At each step the splitting process needs to decide on the feature index j to be used for splitting and the location of the split. For a binary variable there is only one possible split location, but for continuous variables there are many possible split locations. The goal is to find the split that creates the most homogeneous subsets. In the case of regression trees, the best split is the one that minimizes the sum of squared errors. In the case of classification trees, the best split is the one that minimizes the Gini impurity. The Gini impurity is a measure of how homogeneous the subsets are.

Below we'll explore tree-based models using the classic diamonds dataset, which contains prices and attributes for 53,940 diamonds. We'll start with simple decision trees, progress to ensemble methods like random forests and gradient boosting, and develop deep insights into how these algorithms work, when to use them, and how to avoid common pitfalls.

Let's look at the data, which has 10 variables:

Variable	Description	Values
<code>carat</code>	Weight of the diamond	Numeric
<code>cut</code>	Quality of the cut	Fair, Good, Very Good, Premium, Ideal
<code>color</code>	Color of the diamond	J, I, H, G, F, E, D
<code>clarity</code>	Clarity of the diamond	I1, SI2, SI1, VS2, VS1, VVS2, VVS1, IF
<code>depth</code>	Depth of the diamond	Numeric
<code>table</code>	Width of the diamond's table	Numeric

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	62	55	326	4.0	4.0	2.4
0.21	Premium	E	SI1	60	61	326	3.9	3.8	2.3
0.23	Good	E	VS1	57	65	327	4.0	4.1	2.3
0.29	Premium	I	VS2	62	58	334	4.2	4.2	2.6
0.31	Good	J	SI2	63	58	335	4.3	4.3	2.8
0.24	Very Good	J	VVS2	63	57	336	3.9	4.0	2.5

Let's plot price vs carat. Notice the strong non-linear relationship between carat and price. This suggests that log-transformations might help make the relationship linear.

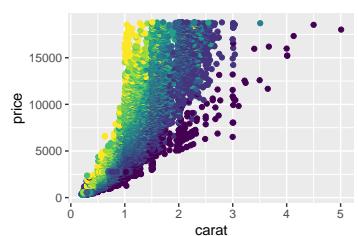


Figure 14.2: Price vs carat

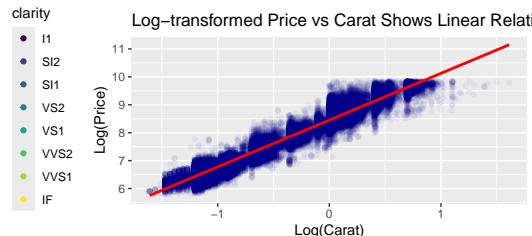
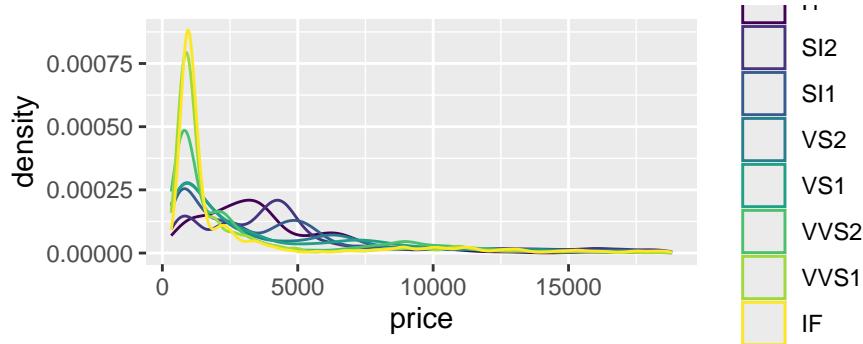


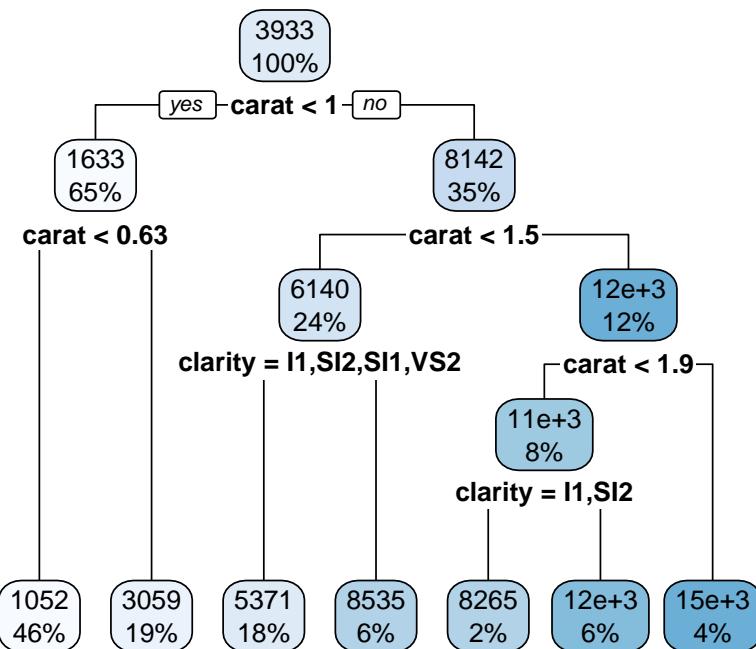
Figure 14.3: Price vs carat

Unlike linear regression, tree models are naturally indifferent to non-linear relationships between predictors and the response. In general, we do not need to transform the variables.

Although carat is the most important factor in determining the price of a diamond, it is not the only factor. We can see that there is a lot of variability in the price of diamonds with the same carat.



Let's start with a simple decision tree using just two predictors to visualize how trees partition the feature space:

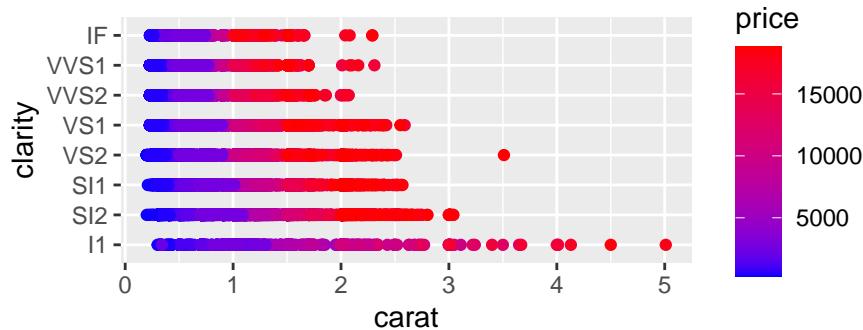


The tree structure reveals several patterns:

1. **Primary split on carat:** The tree first splits at $\text{carat} = 1.05$, confirming carat weight as the strongest price determinant.
2. **Secondary splits on clarity:** Within carat regions, further partitioning on clarity levels shows its secondary predictive value.
3. **Interpretability:** Each leaf shows the predicted price. For example, diamonds with $\text{carat} < 1.05$ and lower clarity (I1, SI2, SI1) have predicted price \$2,847.
4. **Feature interactions:** The hierarchical structure captures how clarity's effect on price depends on carat weight.

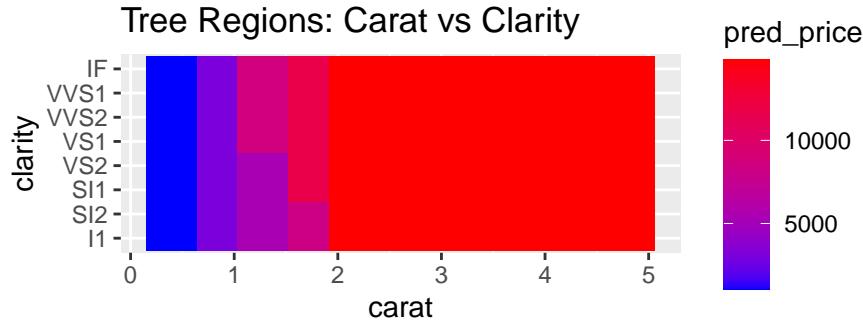
This demonstrates the key advantages of decision trees: non-linear relationships, interpretable rules, and automatic feature interactions.

Let's plot the data.



We can see that for small and large diamonds, the price is consistently low and does not depend much on the clarity. However, at around 1 carat, we see some overlap in the price for different clarity levels. Clarity becomes important at this level.

Now let's plot the data with the tree regions.



The plot above shows the decision tree's prediction regions as colored tiles, where each tile represents a specific combination of carat and clarity values. The color gradient from blue to red indicates the predicted price, with darker red representing higher predicted prices.

Looking at this visualization, we can see several key patterns. The strongest predictor is clearly carat, as evidenced by the vertical bands of similar colors. As carat increases (moving right on the x-axis), the predicted prices generally increase (colors shift from blue to red). The tree captures non-linear patterns that a simple linear model would miss. For example, the rate of price increase with carat is not uniform across all clarity levels. Unlike smooth regression surfaces, the tree creates distinct rectangular regions with sharp boundaries, reflecting the binary splitting nature of decision trees.

14.1 Building a Tree via Recursive Binary Splitting

The prediction using a tree is straightforward. The tree divides the predictor space—that is, the set of possible values for x_1, x_2, \dots, x_p —into J distinct and non-overlapping boxes, R_1, R_2, \dots, R_J . For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j .

$$f(x) = \bar{y}_j, \text{ for } x \in R_j, \text{ where } \bar{y}_j = \text{Average}(y_i \mid x_i \in R_j)$$

The overall goal of building a tree is to find regions that lead to minima of the total Residual Sum of Squares (RSS)

$$\text{RSS} = \sum_{j=1}^J \sum_{i \in R_j} (y_i - \bar{y}_j)^2 \rightarrow \text{minimize}$$

Unfortunately, it is computationally infeasible (NP-hard problem) to consider every possible partition of the feature space into J boxes. We can find a good approximate solution, using top-down approach (the CART algorithm).

It begins with the entire dataset at the “root” node and repeatedly splits the data into two “child” nodes. This process continues recursively on each new node, with the goal of making the resulting groups (nodes) as homogeneous as possible with respect to the target variable, price. At each iteration we decide on which variable j to split and the split point s .

$$R_1(j, s) = \{x \mid x_j < s\} \text{ and } R_2(j, s) = \{x \mid x_j \geq s\},$$

thus, we seek to minimize (in case of regression tree)

$$\min_{j,s} \left[\sum_{i:x_i \in R_1} (y_i - \bar{y}_1)^2 + \sum_{i:x_i \in R_2} (y_i - \bar{y}_2)^2 \right]$$

As a result, every observed input point belongs to a single region.

14.2 Pruning: Taming an Overfit Tree

Now let's discuss how many regions we should have. At one extreme end, we can have n regions, one for each observation. Then the tree model will work similar to the one-nearest neighbor model. At the other end, we can have one big region for the entire input space and then every prediction will be the same (average across observed y 's). Both models can be used but usually the best one is in the middle. The number of regions (branches) controls the complexity of the model. We need to find a good size on the variance-bias scale. A smaller tree with fewer splits (that is, fewer regions R_1, \dots, R_J) might lead to lower variance and better interpretation at the cost of a little bias. Deep trees often suffer from high variance, where slight perturbations in the training data produce vastly different structures, rendering the model unstable.

How do we construct a tree with a “manageable” number of branches? This is accomplished through the steps of forward tree construction and backward pruning. The forward step is a greedy algorithm that begins with a single region and divides it into two. This procedure is repeated until a certain stopping criterion is met. A practical method is to continue building the tree until the Residual Sum of Squares (RSS) plateaus. However, this method can be myopic as an initially unproductive split might be followed by a highly beneficial one, leading to a significant decrease in RSS in subsequent iterations. A more effective strategy is to grow an extensive tree T_0 , and then trim it down

to obtain a subtree. The size of the subtree can be determined using cross-validation. However, be aware that the number of subtrees can be exponential!

Instead of considering all possible sub-trees, we will do cost complexity pruning - also known as weakest link pruning. We consider a sequence of trees indexed by a nonnegative tuning parameter α . For each value of α there corresponds a subtree $T \subset T_0$ such that minimizes

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \bar{y}_m)^2 + \alpha|T|$$

The parameter α balances the complexity of the subtree and its adherence to the training data. When we increment α starting from zero, branches are predictably and sequentially pruned from the tree, making it straightforward to acquire the entire series of subtrees as a function of α . We determine the optimal value $\hat{\alpha}$ through cross-validation. Afterward, we refer back to the complete data set and extract the subtree that corresponds to $\hat{\alpha}$.

14.2.1 Example: Boston Housing Data

To demonstrate pruning and decision boundaries on a standard benchmark, we switch to the Boston Housing dataset. This dataset contains information about housing values in suburbs of Boston. We used it in previous chapters, but here it allows us to easily visualize pruning on a well-known problem.

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.01	18	2.3	0	0.54	6.6	65	4.1	1	296	15	397	5.0	24
0.03	0	7.1	0	0.47	6.4	79	5.0	2	242	18	397	9.1	22
0.03	0	7.1	0	0.47	7.2	61	5.0	2	242	18	393	4.0	35
0.03	0	2.2	0	0.46	7.0	46	6.1	3	222	19	395	2.9	33
0.07	0	2.2	0	0.46	7.2	54	6.1	3	222	19	397	5.3	36
0.03	0	2.2	0	0.46	6.4	59	6.1	3	222	19	394	5.2	29

We will focus on predicting `medv` (median value of owner-occupied homes in \$1000s) using `lstat` (lower status of the population percent) and other variables.

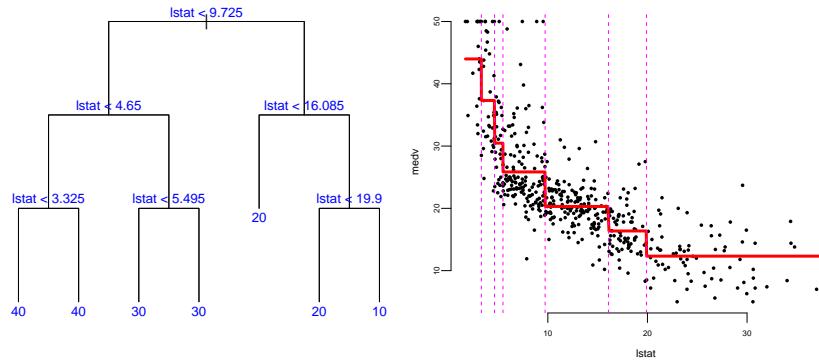
First we build a big tree:

```
library(tree)
# mindev param controls stopping: smaller value = bigger tree
temp <- tree(medv ~ lstat, data = Boston, mindev = .0001)
length(unique(temp$where)) # first big tree size
## 73
```

Then prune it down to one with 7 leaves:

```
boston.tree <- prune.tree(temp, best = 7)
length(unique(boston.tree$where)) # pruned tree size
## 7

text(boston.tree, col = "blue", label = c("yval"), cex = .8)
boston.fit <- predict(boston.tree) # get training fitted values
plot(Boston$lstat, Boston$medv, cex = .5, pch = 16, xlab =
  "lstat", ylab = "medv") # plot data
oo <- order(Boston$lstat)
lines(Boston$lstat[oo], boston.fit[oo], col = "red", lwd = 3) #
  step function fit
cvals <- c(9.725, 4.65, 3.325, 5.495, 16.085, 19.9) # cutpoints
  from tree
for (i in seq_along(cvals)) abline(v = cvals[i], col =
  "magenta", lty = 2) # cutpoints
```



Now let's use more variables. We pick `dis` (weighted mean of distances to five Boston employment centres), `lstat`, and `medv`:

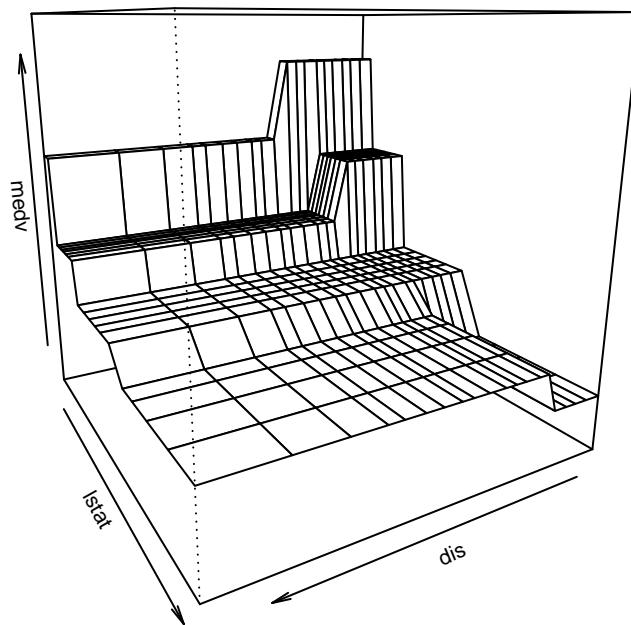
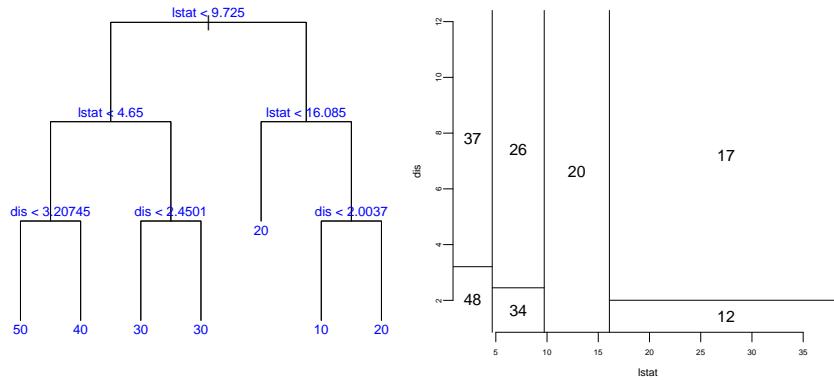
```
## "dis"    "lstat"  "medv"
```

Build the big tree:

```
## 74
```

Then prune it down to one with 7 leaves:

Get predictions on 2d grid and make perspective plot:



Advantages: Decision trees are intuitive, visually interpretable, and handle qualitative predictors without dummy variables.

Disadvantages: The main drawback is instability—deep trees suffer from high variance, where small data perturbations produce vastly different structures.

14.3 Classification Trees

A classification tree operates much like a regression tree. The prediction is made based on the “majority vote”, which means selecting the class that appears most frequently within the region. The process of developing a classification tree is largely the same as that of a regression tree, involving recursive binary splitting. However, instead of using the Residual Sum of Squares (RSS), we use criteria better suited for categorical data.

We start by introducing some notations. Let

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

be the proportion of observations of class k in region m .

The classification is then done by:

$$p_m = \max_k p_{mk}, \quad E_m = 1 - p_m$$

where $k(m) = \arg \max_k p_{mk}$ is the most frequent class in region m . The error rate E_m is simply the proportion of observations in the region that do NOT belong to the majority class.

Then classification prediction is:

$$P(y = k) = \sum_{j=1}^J p_j I(x \in R_j)$$

An alternative method to evaluate the quality of a split in a classification tree is through the use of the Gini Index or Cross-Entropy. Let's consider a scenario where we have an equal number of observations in each class, say 400 in each.

Now, suppose we create a tree that results in two regions: one with a distribution of (300,100) and the other with (100,300). This means that in the first region, 300 observations belong to one class and 100 to the other, and vice versa in the second region. Consider another scenario where we have a different tree that results in two regions with distributions of (200,400) and (200,0).

In both cases, the misclassification rate is 0.25, meaning that 25% of the observations are incorrectly classified. However, the second tree is more desirable. Why is that? The second tree has a region with no misclassifications at all (200,0), which means it is perfectly classifying all observations in that region. This is an ideal situation in classification problems.

This illustrates that while the misclassification rate is a useful metric, it does not always capture the full picture. The Gini Index or Cross-Entropy are preferred for growing trees because they are more sensitive to node purity.

The Gini index:

$$G_m = \sum_{k=1}^K p_{mk}(1 - p_{mk})$$

It measures a variance across the K classes. It takes on a small value if all of the p_{mk} 's are close to zero or one (pure nodes).

An alternative to the Gini index is cross-entropy (a.k.a deviance), given by

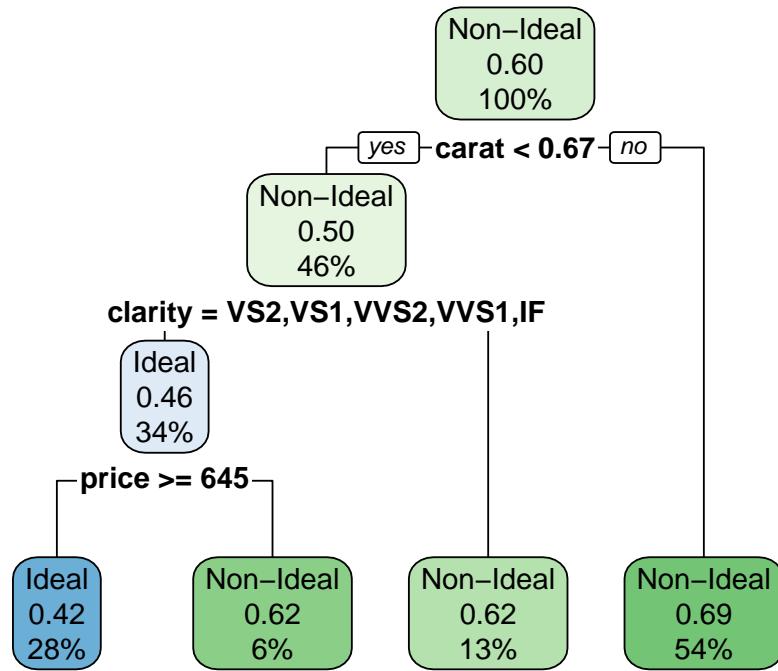
$$D_m = - \sum_{k=1}^K p_{mk} \log p_{mk}$$

It is near zero if the p_{mk} 's are all near zero or near one. Gini index and the cross-entropy led to similar results.

14.3.1 Example: Classifying Diamond Quality

Let's use a classification tree to predict the `cut` of a diamond based on its price and carat. We will try to distinguish "Ideal" cuts from others. To make it a clear binary problem for visualization, let's create a binary variable `is_ideal`.

Classification Tree for Ideal Cut



The tree shows how `clarity` and `carat` effectively separate Ideal cut diamonds from the rest. The nodes display the predicted class and the probability of that class, illustrating how the model estimates class probabilities (p_{mk}) in each region.

14.4 Ensemble Methods

Pruning reduces overfitting by simplifying the tree. Two alternative approaches are **Random Forests** (averaging multiple trees) and **Boosting** (sequentially fitting trees to residuals).

The simple idea behind every ensemble method is that the variance of the average is lower than the variance of individual models (see Section 10.1.3 for a historical perspective on how Galton's ox-weighing experiment illustrates

this principle). Say we have B models $f_1(x), \dots, f_B(x)$ then we combine those

$$f_{avg}(x) = \frac{1}{B} \sum_{b=1}^B f_b(x)$$

Combining models helps fight overfitting. On the negative side, it is harder to interpret these ensembles compared to a single decision tree.

14.4.1 Bagging

In the **bagging** approach, we treat the sample as if it were the population and then take iid draws. That is, you sample with replacement so that you can get the same original sample value more than once in a bootstrap sample.

To bootstrap aggregate (Bag) we:

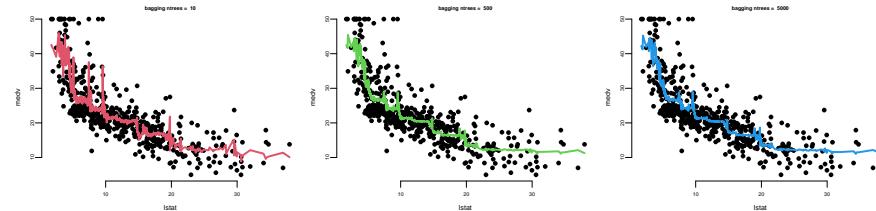
- Take B bootstrap samples from the training data, each of the same size as the training data.
- Fit a large tree to each bootstrap sample (we know how to do this fast!). This will give us B trees.
- Combine the results from each of the B trees to get an overall prediction.

When the target variable y is numeric, the bagging process is straightforward: the final prediction is simply the average. When y is categorical, we use a voting system or average the predicted probabilities.

Let's experiment with the number of trees in the model using the Boston dataset again. We calculate the mean squared error for each number of trees.

```
## Mean Squared Error with 10 trees: 31
## Mean Squared Error with 500 trees: 29
## Mean Squared Error with 5000 trees: 29
```

Let's plot the results



- With 10 trees our fit is too jumbly (high variance).
- With 1,000 and 5,000 trees the fit is smooth and very similar.
- Note that although our method is based on multiple trees (average over) so we no longer have a simple step function!

14.4.2 Random Forest

In the bagging technique, models can become correlated, which prevents the achievement of a $1/B$ reduction in variance. This happens because most, if not all, of the trees will use the most influential predictor in the top split. As a result, bagged trees tend to look very similar to each other.

Random Forests, on the other hand, introduce an element of randomness that helps to decorrelate the trees. Instead of considering all p predictors for a split, a random sample of m predictors is chosen as split candidates. This subset of predictors is different for each split.

The number of predictors considered at each split, m , is typically chosen to be the square root of the total number of predictors, p for classification, or $p/3$ for regression.

One of the “interpretation” tools that comes with ensemble models is importance ranking: the total amount that the deviance (loss) is decreased due to splits over a given predictor, averaged over all trees.

```
rf.boston <- randomForest(medv ~ ., data = Boston, mtry = 4,
                             importance = TRUE, ntree = 50)
varImpPlot(rf.boston, pch = 21, bg = "lightblue", main =
            "Variable Importance")
```

Variable Importance

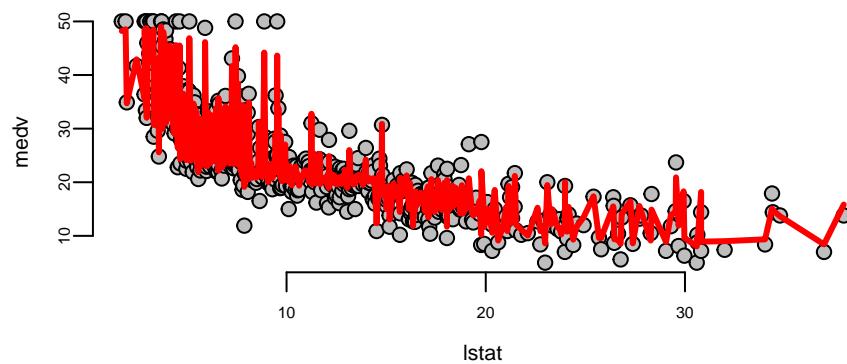
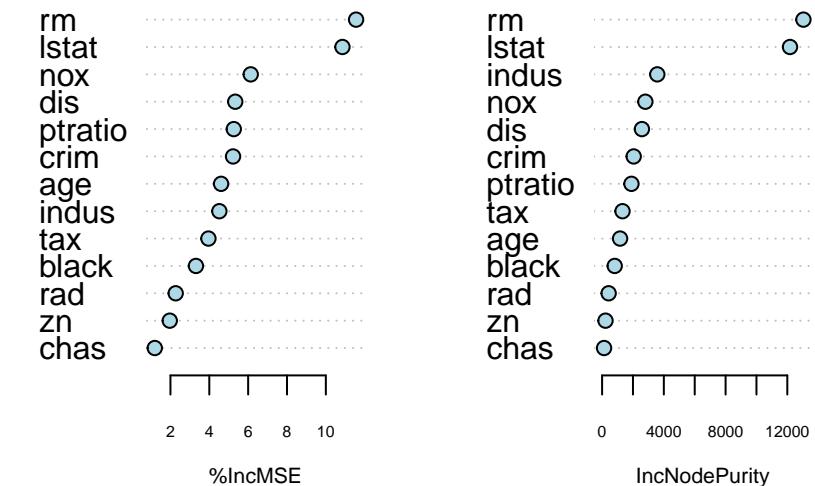


Figure 14.4: Random Forest Fit

14.4.3 Boosting

Boosting, like Random Forests, is a method that combines multiple trees to create a more powerful predictive model. However, the approach it takes is

quite distinct.

Here's how Boosting works:

1. Initially, a single decision tree is fitted to the data. This initial tree is intentionally made weak.
2. We then examine the residuals, which represent the portion of the target variable y not explained by the weak tree.
3. A new tree is then fitted to these residuals, essentially trying to predict the error of the first tree.
4. This new tree is also “weakened” or “shrunk” (multiplied by a learning rate λ). The prediction from this tree is then added to the prediction of the previous trees.
5. This process is repeated iteratively. The final model is the sum of all these “shrunk” trees.

The key idea behind Boosting is to iteratively improve the model by focusing on the parts of the data that the current model is not explaining well (the residuals).

Mathematically, we want to minimize a loss function \mathcal{L} . For regression, we might take $\mathcal{L}(y_i, \theta_i) = (y_i - \theta_i)^2$. We solve:

$$\underset{\beta \in R^B}{\text{minimize}} \sum_{i=1}^n \mathcal{L} \left(y_i, \sum_{b=1}^B \beta_j \cdot T_b(x_i) \right)$$

Gradient boosting acts like gradient descent in function space. Start with initial model, e.g., fit a single tree $\theta^{(0)} = T_0$. Repeat: - Evaluate gradient g at latest prediction $\theta^{(k-1)}$,

$$g_i = \left[\frac{\partial \mathcal{L}(y_i, \theta_i)}{\partial \theta_i} \right] \Big|_{\theta_i=\theta_i^{(k-1)}}, \quad i = 1, \dots, n$$

- Find a tree T_k that is close to $-g$, i.e., T_k solves

$$\underset{\text{trees } T}{\text{minimize}} \sum_{i=1}^n (-g_i - T(x_i))^2$$

- Update our prediction:

$$\theta^{(k)} = \theta^{(k-1)} + \lambda \cdot T_k$$

Algorithm: 1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in training set. 2. For $b = 1, 2, \dots, B$, repeat: (a) Fit a tree \hat{f}^b with d splits (interactions) to the training data (X, r) . (b) Update \hat{f} by adding a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

(c) Update the residuals:

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model:

$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x)$$

The parameter λ is the **learning rate** or *shrinkage* parameter. By multiplying the new tree's contribution by a small number (e.g., 0.01 or 0.1), we prevent the model from adapting too quickly to outliers or noise.

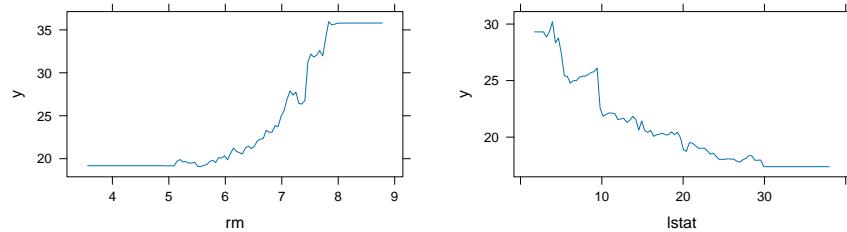
Here are some boosting fits where we vary the number of trees, but fix the depth at 2 (suitable with 1 x) and shrinkage = λ at .2.

```
library(gbm)
boost.boston <- gbm(medv ~.,
  data = Boston, distribution = "gaussian",
  n.trees = 5000, interaction.depth = 4
)
yhat.boost <- predict(boost.boston, newdata = Boston, n.trees =
  ↪ 5000)
mean((yhat.boost - Boston$medv)^2)
## 0.0004
```

summary(boost.boston, plotit = FALSE)

	var	rel.inf
lstat	lstat	36.32
rm	rm	30.98
dis	dis	7.63
crim	crim	5.09
nox	nox	4.63
age	age	4.50
black	black	3.45
ptratio	ptratio	3.11
tax	tax	1.74
rad	rad	1.17
indus	indus	0.87
chas	chas	0.39
zn	zn	0.13

Boosting vs Random Forests: * Boosting often provides better accuracy by correcting specific errors (residuals), but is more prone to overfitting and



noise. It requires careful tuning of λ and B . * **Random Forests** are more robust “out of the box”, easier to parallelize (trees are independent), and less prone to overfitting, but might not reach the same peak accuracy as a well-tuned boosting model.

14.5 BART for causal inference

Estimating the causal effect of an intervention, such as a new drug, a marketing campaign, or a public policy, is a central goal across science and industry. While the gold standard for causal inference is the Randomized Controlled Trial (RCT), it is often infeasible, unethical, or too expensive to conduct. Researchers must therefore turn to observational data, where the assignment of treatment is not controlled by the investigator. This introduces a fundamental challenge: individuals who receive the treatment may be systematically different from those who do not, a problem known as confounding. Separating the true effect of the treatment from these pre-existing differences is the primary task of causal inference from observational data.

To formalize causal questions, we rely on the Rubin Causal Model (RCM), also known as the potential outcomes framework. For a binary treatment Z (where $Z_i = 1$ if individual i receives the treatment and $Z_i = 0$ otherwise), we posit that each individual i has two potential outcomes: * $Y_i(1)$: The outcome that would be observed if individual i were exposed to the treatment. * $Y_i(0)$: The outcome that would be observed if individual i were exposed to the control (no treatment).

This framework leads directly to what Holland (1986) termed the “fundamental problem of causal inference”: for any given individual, we can only ever observe one of these two potential outcomes. The outcome we do not see is the counterfactual. Causal inference can thus be viewed as a missing data problem, where the goal is to estimate the values of the unobserved potential outcomes.

From this foundation, we can define several key causal quantities, or estimands:

- Individual Treatment Effect (ITE): The effect for a single individual, defined as

$$\tau_i = Y_i(1) - Y_i(0).$$

This is typically unobservable.

- Average Treatment Effect (ATE): The average effect across the entire population,

$$\tau_{ATE} = E(Y(1) - Y(0)).$$

This is often the primary estimand of interest for broad policy questions.

- Average Treatment Effect on the Treated (ATT): The average effect for those who actually received the treatment,

$$\tau_{ATT} = E(Y(1) - Y(0) | Z = 1).$$

- Conditional Average Treatment Effect (CATE): The average effect for a subpopulation defined by a set of covariates $X = x$,

$$\tau(x) = E(Y(1) - Y(0) | X = x).$$

Understanding the CATE allows for the exploration of treatment effect heterogeneity.

To estimate these causal estimands from observational data, we must rely on a set of critical, untestable assumptions that connect the observed data to the unobserved potential outcomes. These are known as identification assumptions.

1. Stable Unit Treatment Value Assumption (SUTVA): This assumption has two parts. First, it assumes there is no interference between units, meaning one individual's treatment status does not affect another's outcome. Second, it assumes there are no hidden variations of the treatment; the treatment assigned to one individual is the same as the treatment assigned to any other.
2. Ignorability (or Unconfoundedness): This is the most crucial assumption. It states that, conditional on a set of observed pre-treatment covariates X , treatment assignment Z is independent of the potential outcomes:

$$(Y(0), Y(1)) \perp Z | X$$

- . In essence, it assumes that we have measured all the common causes of both treatment selection and the outcome. If this holds, then within any stratum defined by the covariates X , the treatment assignment is “as-if” random.

3. Positivity (or Overlap/Common Support): This assumption requires that for any set of covariate values x present in the population, there is a non-zero probability of being in either the treatment or the control group: $0 < P(Z = 1 | X = x) < 1$. This ensures that we can find both treated and control individuals with similar characteristics, making comparison meaningful and avoiding extrapolation to regions with no data.

To demonstrate the application of Bayesian methods to this challenge, we use the famous Lalonde dataset, a canonical benchmark in the causal inference literature. The dataset addresses a real-world policy question: evaluating the effectiveness of the National Supported Work (NSW) Demonstration, a federally funded job training program implemented in the US from 1975-1979. The program was designed to help individuals facing significant social and economic barriers (e.g., former drug addicts, ex-convicts, high school dropouts) improve their labor market prospects. The treatment (*treat*) is participation in this program, and the primary outcome (*re78*) is the individual's real earnings in 1978, after the program.

The historical importance of this dataset stems from Robert Lalonde's 1986 paper, which delivered a powerful critique of the non-experimental methods used at the time. Lalonde started with data from an actual RCT, which provided an unbiased estimate of the program's effect. He then took the treated group from the experiment but replaced the experimental control group with a non-experimental comparison group drawn from large public surveys—the Panel Study of Income Dynamics (PSID) and the Current Population Survey (CPS). He showed that the standard econometric models of the era failed to replicate the experimental benchmark when applied to this new, confounded dataset, casting serious doubt on their reliability for policy evaluation. Our task is to see if a modern, flexible Bayesian method—Bayesian Additive Regression Trees (BART)—can succeed where these earlier methods failed.

The challenge posed by the Lalonde dataset becomes immediately apparent when we examine the pre-treatment characteristics of the treated group versus the non-experimental control group. A naive comparison of their 1978 earnings would be deeply misleading because the groups were profoundly different before the program even began. Table 14.5 illustrates this imbalance for key covariates, including age, education, race, marital status, and earnings in the years prior to the intervention (1974 and 1975).

The Standardized Mean Difference (SMD) provides a scale-free measure of the difference between the group means. A common rule of thumb suggests that an absolute SMD greater than 0.1 indicates a potentially meaningful imbalance. As the table shows, the groups differ substantially on nearly every measured characteristic. The treated individuals were younger, less educated, more likely to be from minority groups, and had drastically lower earnings in the years before the program. This severe selection bias is precisely what

makes the Lalonde dataset such a difficult and important test case for causal inference methods. Any credible method must be able to adjust for these vast pre-existing differences to isolate the true causal effect of the job training program.

Table 14.5: Covariate Balance in the Lalonde Non-Experimental Dataset.
Note: Data corresponds to the widely used Dehejia and Wahba (1999) sample of the Lalonde dataset. Standardized Mean Difference is calculated as the difference in means divided by the pooled standard deviation.

Covariate	Treated Mean	Control Mean	Std. Mean Diff.
Age (years)	25.82	28.04	-0.31
Education (years)	10.35	10.23	0.06
Black (indicator)	0.84	0.20	1.84
Hispanic (indicator)	0.06	0.14	-0.32
Married (indicator)	0.19	0.51	-0.81
No Degree (indicator)	0.71	0.60	0.25
Earnings 1974	2095.57	5630.71	-0.63
Earnings 1975	1532.06	5205.52	-0.65

To address the challenge of confounding, we need a method that can flexibly model the relationship between the outcome, the treatment, and the many covariates shown to be imbalanced. Bayesian Additive Regression Trees (BART) is a powerful non-parametric machine learning algorithm that is exceptionally well-suited for this task. It combines the predictive power of ensemble methods with a rigorous Bayesian framework for regularization and uncertainty quantification.

At its core, BART models the expected value of an outcome Y as a sum of many individual regression trees. For a set of predictors x , the model is:

$$Y = \sum_{j=1}^m g(x; T_j, M_j) + \epsilon, \quad \text{where } \epsilon \sim N(0, \sigma^2)$$

Here, m is the number of trees in the ensemble (typically around 200), and each function $g(x; T_j, M_j)$ represents a single regression tree. The structure of the tree is denoted by T_j , and M_j is the set of parameter values in its terminal nodes (or leaves).

Crucially, each individual tree is designed to be a “weak learner”. It is kept shallow and simple, meaning it explains only a small fraction of the variation in the outcome. The final, powerful prediction comes from summing up the contributions of all these simple components. This sum-of-trees structure allows BART to automatically capture very complex relationships, including

high-order interactions and non-linearities, without the user needing to specify them in advance. For example, an interaction between age and education is implicitly modeled if a tree splits on education within a branch that has already been split on age. This flexibility is a major advantage in observational studies where the true functional form of the relationship between the outcome and the confounders is unknown.

In most machine learning algorithms, overfitting is controlled through techniques like cross-validation or complexity penalties. BART, being a fully Bayesian method, achieves this through a carefully specified set of regularization priors. These priors are designed to keep each tree simple and prevent any single tree from dominating the overall fit.

The key priors are:

1. Prior on Tree Structure: This prior strongly encourages shallow trees. It is defined by a rule governing the probability that a node at a certain depth d will be split further. This probability is typically modeled as

$$p(T_j) = \alpha(1 + d)^{-\beta},$$

where $\alpha \in (0, 1)$ and $\beta \geq 0$ are hyperparameters. Setting β to a value like 2 ensures that the probability of splitting decreases rapidly with depth, keeping the trees small.

2. Prior on Terminal Node Parameters: After the response variable Y is centered and scaled, the values μ_{jk} in the terminal nodes of each tree are given a Normal prior,

$$\mu_{jk} \sim N(0, \sigma_\mu^2).$$

This prior shrinks the predictions within each leaf towards zero. Because the final prediction is a sum over m trees, this shrinkage ensures that the contribution of each individual tree is small.

3. Prior on Error Variance: The residual variance σ^2 is typically given a conjugate Inverse-Gamma prior. This prior is usually chosen to be weakly informative, allowing the data to dominate the posterior estimate of the noise level, but it still constrains the variance to be reasonable.

Together, these priors act as a sophisticated regularization mechanism that allows BART to fit complex functions while being highly resistant to overfitting.

BART models are fit using a Markov chain Monte Carlo (MCMC) algorithm, specifically a form of Gibbs sampler known as Bayesian backfitting. The algorithm does not find a single “best” model. Instead, it generates thousands of samples from the joint posterior distribution of all model parameters: $p(T_1, \dots, T_m, M_1, \dots, M_m, \sigma | Y, X)$.

The fitting process works iteratively:

1. Initialize all m trees and σ .
2. For each tree j from 1 to m :
 - a. Calculate the “partial residual” by subtracting the predictions of all other trees from the outcome:

$$R_j = Y - \sum_{k \neq j} g(x; T_k, M_k)$$

- b. Draw a new tree structure T_j and its leaf parameters M_j from their posterior distribution conditional on this partial residual,

$$p(T_j, M_j | R_j, \sigma).$$

3. After iterating through all trees, draw a new value for σ from its posterior conditional on the current residuals.
4. Repeat steps 2 and 3 for thousands of iterations.

The output of this process is not one set of trees, but a collection of (e.g., 5000) sets of trees, where each set represents a plausible regression function drawn from the posterior distribution. This collection of draws is the key to quantifying uncertainty in a Bayesian way.

The power of BART for causal inference lies in how it leverages the full posterior distribution to estimate counterfactuals. The strategy aligns perfectly with the Bayesian view of causal inference as a missing data problem, as articulated by Rubin (1978).

The standard approach for causal inference with BART is to model the outcome Y as a function of both the covariates X and the treatment indicator Z . The model learns a single, flexible response surface:

$$\text{E}(Y | X, Z) = f(X, Z)$$

Here, the treatment Z is included as if it were “just another covariate” in the set of predictors fed to the BART algorithm. The model is free to discover how the effect of Z varies with X through the tree-splitting process. The Conditional Average Treatment Effect (CATE) is then simply the difference in the predictions from this learned function:

$$\tau(x) = f(x, Z = 1) - f(x, Z = 0)$$

The core of the estimation process is a predictive step that is repeated for each draw from the MCMC sampler. Suppose the MCMC algorithm has produced S posterior draws of the function f . For each draw $s = 1, \dots, S$:

1. We take the full dataset of n individuals with their observed covariates X .
2. We create two hypothetical, or counterfactual, datasets:
 - Treated World: The observed covariates X for all n individuals, but with the treatment indicator set to $Z = 1$ for everyone.
 - Control World: The observed covariates X for all n individuals, but with the treatment indicator set to $Z = 0$ for everyone.
3. Using the fitted BART model corresponding to posterior draw s (i.e., $f^{(s)}$), we predict the outcome for every individual under both scenarios. This gives us a full set of posterior predictive draws for the potential outcomes: $\tilde{Y}_i(1)^{(s)}$ and $\tilde{Y}_i(0)^{(s)}$ for each individual i .

This process is a direct implementation of the missing data analogy. For an individual i who was actually treated ($Z_i = 1$), their observed outcome Y_i is their potential outcome $Y_i(1)$. The BART model provides a posterior predictive draw for their missing counterfactual outcome, $\tilde{Y}_i(0)^{(s)}$. Conversely, for a control subject, we use the model to predict their missing $\tilde{Y}_i(1)^{(s)}$.

Once we have the posterior draws of the potential outcomes for every individual at each MCMC iteration, we can compute a posterior draw for any causal estimand of interest. For example, at each iteration s :

- ITE draw:

$$\tau_i^{(s)} = \tilde{Y}_i(1)^{(s)} - \tilde{Y}_i(0)^{(s)}$$

- ATE draw:

$$\tau_{ATE}^{(s)} = \frac{1}{n} \sum_{i=1}^n \tau_i^{(s)}$$

By collecting these values across all S MCMC iterations, we obtain

$$\{\tau_{ATE}^{(1)}, \tau_{ATE}^{(2)}, \dots, \tau_{ATE}^{(S)}\}.$$

This set is a Monte Carlo approximation of the entire posterior distribution of the Average Treatment Effect.

This is a profoundly powerful result. Instead of a single point estimate and a standard error, the Bayesian approach yields a full probability distribution for the unknown causal effect. From this posterior distribution, we can easily calculate a posterior mean (our best point estimate) and a 95% credible interval. Unlike a frequentist confidence interval, the Bayesian credible interval has a direct and intuitive probabilistic interpretation: given our data and model, there is a 95% probability that the true value of the ATE lies within this range. This propagation of uncertainty from the model parameters all the way to the final causal estimate is a hallmark of the Bayesian approach.

We now apply this framework to the Lalonde dataset to estimate the causal effect of the NSW job training program on 1978 earnings.

The analysis is streamlined by using the `bartCause` package in R, which is specifically designed for causal inference with BART. The package provides a wrapper around the core `dbarts` implementation, simplifying the process of fitting the model and generating counterfactuals. A typical function call would look like this:

```
# Load the package and data
library(bartCause)
data(lalonde)

# Define confounders
confounders <- c("age", "educ", "black", "hisp", "married",
                 "nodegr", "re74", "re75")

# Fit the BART model
fit <- bartc(
  response = lalonde$re78,
  treatment = lalonde$treat,
  confounders = lalonde[, confounders],
  estimand = "ate",
  commonSup.rule = "sd" # Rule to handle poor overlap
)
```

In this call, we specify the outcome (`re78`), the binary treatment (`treat`), and the matrix of pre-treatment confounders. We set `estimand = ate` to target the Average Treatment Effect.

Before interpreting the causal estimates, it is essential to perform MCMC diagnostics to ensure the algorithm has converged to a stable posterior distribution. The `bartCause` package provides plotting functions for this purpose. Trace plots for key parameters, such as the posterior draws of the ATE and the residual standard deviation (σ), should be examined. These plots should show the chains mixing well and exploring a consistent region of the parameter space, without long-term drifts or stuck periods, indicating that the sampler has converged.

The primary result can be obtained by calling `summary(fit)`. This provides the posterior mean of the ATE, which serves as our point estimate, along with a 95% credible interval. For a richer view, we can plot the entire posterior distribution of the ATE, which visualizes our uncertainty about the treatment effect.

The true power of this result is seen when placed in the context of other estimates, as shown in Table 14.6. The naive difference in means between the

treated and control groups in the non-experimental data is large and negative, a direct consequence of the severe confounding. The experimental benchmark from the original RCT for this subset of treated individuals is an earnings gain of approximately \$886. The BART estimate, after adjusting for the observed confounders, is remarkably close to this benchmark. This result demonstrates that a flexible, non-parametric Bayesian model like BART can successfully overcome the severe selection bias that plagued earlier econometric methods.

Table 14.6: Comparison of ATE Estimates for the NSW Program. Note: Estimates are for the non-experimental Lalonde sample (treated units from NSW, control units from PSID). The experimental benchmark is the difference-in-means estimate from the randomized trial for the same treated units. Uncertainty for BART is the posterior standard deviation

Method	ATE Estimate	Uncertainty (Std. Dev. / Interval)
Experimental Benchmark	886.3	-277.37
Naive Difference-in-Means	-8492.24	-633.91
Propensity Score Matching	1079.13	-158.59
Double Machine Learning	370.94	-394.68
Causal BART	818.79	-184.46

While the ATE provides a useful summary, it can mask important variations in how the treatment affects different people. A policy might be beneficial on average but ineffective or even harmful for certain subgroups. A key advantage of BART is its ability to move beyond the average and explore this Heterogeneous Treatment Effect (HTE), which is critical for developing more targeted and effective policies.

Estimating HTE allows us to answer questions like: “For whom does this program work best?” or “Are there individuals for whom the program is detrimental?” In settings with limited resources, this information is vital for allocating the intervention to those most likely to benefit. The flexibility of BART, which does not assume a constant treatment effect, makes it an ideal tool for this task.

Because BART provides a posterior predictive distribution of potential outcomes for every individual in the dataset, we can estimate an Individual Conditional Average Treatment Effect (ICATE) for each person. By plotting a histogram of the posterior means of these ICATEs, we can visualize the distribution of effects across the sample. This reveals whether the effect is consistent

for everyone or if there is substantial variation, with some individuals benefiting much more than others.

To understand what drives this heterogeneity, we can examine how the estimated CATE varies as a function of key pre-treatment covariates. These relationships are often visualized using partial dependence plots. For the Lalonde data, such analyses have revealed that the effect of the job training program is not constant but varies non-linearly with characteristics like age and pre-treatment income (`re74`). For instance, the program's benefit might increase with age up to a certain point and then decline, or it might be most effective for individuals with low-to-moderate prior earnings but less so for those with very low or higher earnings. These are nuanced, data-driven insights that would be completely missed by a standard linear regression model that only estimates a single average effect.

A subtle but important issue can arise when using flexible regularized models like BART for causal inference in the presence of strong confounding, as is the case here. The regularization priors, which are designed to prevent overfitting, can shrink the estimated effects of the confounders towards zero. Because the treatment Z is highly correlated with these confounders, the model may mistakenly attribute some of the effect of the confounders to the treatment, leading to a bias known as Regularization-Induced Confounding (RIC).

A powerful solution, proposed by Hahn, Murray, and Carvalho (2020), is to first estimate the propensity score, $\pi(x) = P(Z = 1 | X)$, which is the probability of receiving treatment given the covariates X . This score serves as a one-dimensional summary of all confounding information. This estimated propensity score is then included as an additional predictor in the BART outcome model. By providing this confounding summary directly to the model, we help the BART algorithm differentiate between the prognostic effects of the covariates (captured by $\pi(x)$) and the causal effect of the treatment Z , thereby mitigating RIC. This “ps-BART” approach is considered state-of-the-art and is easily implemented in the `bartCause` package by setting the argument `p.scoreAsCovariate = TRUE`.

14.5.1 BART versus Propensity Score Matching (PSM)

BART is one of several methods for causal inference from observational data. It is instructive to compare its philosophy with that of another widely used technique: Propensity Score Matching (PSM). BART and PSM represent two different philosophies for tackling confounding. Propensity Score Matching (PSM): This approach focuses on the design of the study. The goal is to use the observed data to construct a new sample in which the treatment and control groups are balanced on their observed covariates, thereby mimicking the properties of an RCT. The propensity score is the central tool used to

achieve this balance. The analysis of the outcome is then performed on this newly created, “balanced” dataset.

BART focuses on the analysis stage. The goal is to build a highly flexible and accurate predictive model for the outcome that explicitly includes the treatment and confounders, $E(Y | X, Z)$. It uses the full dataset and relies on the model’s ability to correctly adjust for the confounding variables to isolate the causal effect.

Each approach has its own set of advantages and disadvantages. PSM is often praised for its transparency; one can assess the quality of the covariate balance achieved by the matching procedure before ever looking at the outcome variable, reducing the risk of “p-hacking” or specification searching. However, PSM can be inefficient, as it often requires discarding a significant portion of the control group that does not have good matches in the treated group (i.e., poor overlap). It can also suffer from residual confounding if the matches are not sufficiently close. BART, on the other hand, is highly efficient as it uses all available data. Its main strengths are its flexibility in capturing unknown functional forms and interactions, its ability to easily estimate heterogeneous effects, and its principled framework for uncertainty quantification. Its primary weakness is that it can be perceived as a “black box” if not diagnosed carefully. Its validity, like all modeling approaches, depends on the untestable ignorability assumption, and as discussed, it can be susceptible to regularization-induced confounding if not applied with care.

In modern practice, the line between these two philosophies is blurring. It is now common to see them used in conjunction. For example, many practitioners use flexible machine learning models, including BART itself, to estimate the propensity scores used for matching or weighting, which can improve the quality of the covariate balance over simpler logistic regression models. Conversely, the state-of-the-art application of BART for causal inference (ps-BART) incorporates the propensity score directly into the outcome model. This convergence reflects a mature understanding that both balancing the data structure and flexibly modeling the outcome are complementary and powerful tools for robust causal inference.

Feature	Propensity Score Matching (PSM)	Bayesian Additive Regression Trees (BART)
Primary Goal	Create balanced treatment/control groups (Design)	Flexibly model the outcome-covariate relationship (Analysis)
Use of Data	Often discards unmatched units, reducing sample size	Uses all available data

Feature	Propensity Score Matching (PSM)	Bayesian Additive Regression Trees (BART)
Confounding Control	Achieved by balancing covariates via matching/weighting	Achieved by conditioning on covariates in a flexible model
Key Assumption	Correct specification of the propensity score model	Correct specification of the outcome model (though BART is very flexible)
Treatment Effect	Primarily estimates ATT; ATE can be harder to estimate	Easily estimates ATE, ATT, and CATE/HTE
Uncertainty	Often requires bootstrapping for standard errors	Provides full posterior distributions and credible intervals naturally
Flexibility	Limited by the PS model; main effect is assumed constant after matching	Highly flexible; automatically models non-linearities and interactions

:: Conceptual Comparison of BART and Propensity Score Matching

This example shows that BART, a flexible non-parametric method, can successfully adjust for severe confounding and recover a causal estimate that is remarkably close to the experimental benchmark, a feat that eluded many of the methods available when Lalonde first published his critique. It is crucial to remember that BART is not a panacea. Its validity, like that of any non-experimental method, rests on the untestable assumption of ignorability—that we have measured and adjusted for all relevant confounding variables.

14.5.2 Conclusion

Randomization remains the cleanest route to causal identification, but it is not always feasible. When randomization is absent, modern Bayesian methods like BART offer a principled path forward: they combine flexible non-parametric modeling with uncertainty quantification and can, under the right assumptions, recover causal effects that approach the quality of experimental benchmarks. The broader narrative on experiments, randomization, and when we must rely on observational data is developed in Chapter 5.

14.6 Appendix: Theoretical Foundations

14.6.1 Why Ensembles Work: A Geometric Perspective

Why does combining multiple weak learners improve performance? The answer lies in the **bias-variance tradeoff** and the geometry of high-dimensional spaces.

14.6.2 Smoothing and Variance Reduction

In essence, individual decision trees are low-bias but high-variance estimators. They can capture complex patterns (low bias) but are sensitive to noise (high variance). When we average many such trees, as in Bagging or Random Forests, we are effectively performing **smoothing**.

Quantitatively, if we have N uncorrelated predictors f_1, \dots, f_N , the variance of their average is:

$$\text{Var} \left(\frac{1}{N} \sum_{i=1}^N f_i \right) = \frac{1}{N} \text{Var}(f_i)$$

Averaging N uncorrelated models reduces the variance by a factor of N . This is the **$1/N$ rule**. In practice, the trees in a forest are correlated, so the reduction isn't quite $1/N$, but the principle holds: the more diverse (uncorrelated) the trees, the more variance reduction we get. This is why Random Forests Randomly select features at each split—to force the trees to be different.

14.6.3 The Problem of High Dimensions

One might ask: “Why not just use K-Nearest Neighbors (KNN)? It also averages local points.” The problem is the **curse of dimensionality**. In high-dimensional feature spaces, data points become incredibly sparse.

Consider a 50-dimensional sphere. As shown in Figure 14.5, if we sample points uniformly, almost all of them will reside near the “crust” or surface of the sphere, not in the center.

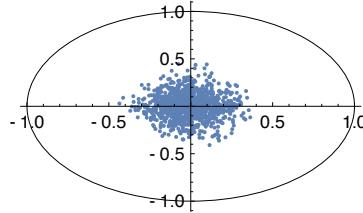


Figure 14.5

This phenomenon means that in high dimensions, “local neighbors” are not actually close to you—they are far away on the other side of the space. A standard KNN algorithm fails because it averages points that aren’t truly similar.

14.6.4 Trees as “Adaptive” Nearest Neighbors

Decision trees solve this by defining “neighbors” differently. Instead of using a fixed distance metric (like Euclidean distance), trees define a neighborhood as a rectangular box (or **cylindrical region**) learned from the data (Figure 14.6).

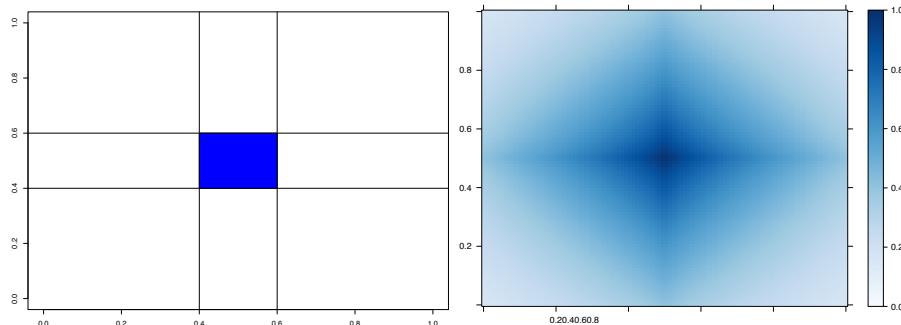


Figure 14.6: Cylindrical kernels for trees (left) and random forests (right).

Constructing the regions is fundamental to reducing the curse of dimensionality. It is useful to imagine a very large dataset, e.g., 100k images, and think about how a new image’s input coordinates, X , are “neighbors” to data points in the training set. Our predictor will then be a smart conditional average of the observed outputs, Y , for our neighbors. When p is large, spheres (L^2 balls or Gaussian kernels) are terrible: either no points or all points are “neighbors” of the new input variable. Trees are good as not too many “neighbors”.

To illustrate the problem further, Figure 14.5 below shows the 2D image of 1000 uniform samples from a 50-dimensional ball B_{50} . The image is calculated as $w^T Y$, where $w = (1, 1, 0, \dots, 0)$ and $Y \sim U(B_{50})$. Samples are centered around the equators and none of the samples fall close to the boundary of the set.

As dimensionality of the space grows, the variance of the marginal distribution goes to zero. We can empirically see it from Figure 14.7, which shows histogram of 1D image of uniform sample from balls of different dimensionality, i.e. $e_1^T Y$, where $e_1 = (1, 0, \dots, 0)$.

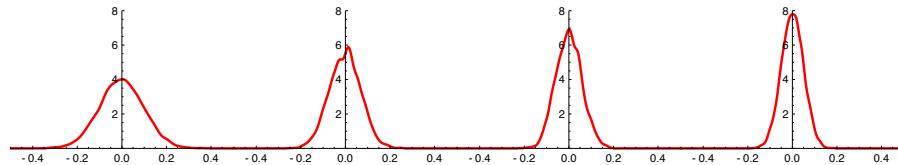


Figure 14.7: Histogram of marginal distribution of $Y \sim U(B_p)$ for different dimensions p (x-axis).

Similar central limit results were known to Maxwell who showed that random variable $w^T Y$ is close to standard normal, when $Y \sim U(B_p)$, p is large, and w is a unit vector (lies on the boundary of the ball). For the history of this fact, see Persi Diaconis and Freedman (1987). More general results in this direction were obtained in Klartag (2007). Further, Milman and Schechtman (2009) presents many analytical and geometrical results for finite dimensional normed spaces, as the dimension grows to infinity.

Deep learning improves on this by performing a sequence of GLM-like transformations; effectively, DL learns a distributed partition of the input space. Specifically, suppose that we have K partitions. Then the DL predictor takes the form of a weighted average, or in the case of classification, a soft-max of the weighted average of observations in this partition. Given a new high-dimensional input X_{new} , many deep learners are an average of learners obtained by our hyperplane decomposition. Generically, we have

$$\hat{Y}(X) = \sum_{k \in K} w_k(X) \hat{Y}_k(X),$$

where w_k are the weights learned in region k , and $w_k(X)$ is an indicator of the region with appropriate weighting given the training data. The weight w_k also indicates which partition the new X_{new} lies in.

The use of pooling (a.k.a. averaging) of multiple predictors is commonplace in machine learning. Ensemble methods (a.k.a. some form of clever conditional averaging) are prevalent in high dimensions. One reason for these procedures

is that it is relatively easy to find unbiased predictors, with the caveat that they have large variances due to dimensionality. The following result on exchangeability (Kingman, 1975) shows that we can simply use the $1/N$ -rule and average to reduce risk. Specifically, suppose that we have K exchangeable, $\mathbb{E}(\hat{Y}_i) = \mathbb{E}(\hat{Y}_{\pi(i)})$, predictors

$$\hat{Y} = (\hat{Y}_1, \dots, \hat{Y}_K)$$

Find w to attain $\operatorname{argmin}_W El(Y, w^T \hat{Y})$ where l convex in the second argument;

$$\begin{aligned} El(Y, w^T \hat{Y}) &= \frac{1}{K!} \sum_{\pi} El(Y, w^T \hat{Y}) \geq El \left(Y, \frac{1}{K!} \sum_{\pi} w_{\pi}^T \hat{Y} \right) \\ &= El \left(Y, (1/K) \iota^T \hat{Y} \right) \end{aligned}$$

where $\iota = (1, \dots, 1)$. Hence, the randomized multiple predictor with weights $w = (1/K)\iota$ provides close to optimal Bayes predictive performance. We now turn to algorithmic issues.

An alternative approach is to perform Bayesian model selection. Here we calculate the optimal Bayes weight for each predictor in accordance with Bayes Rule. We formalize the gains in Classification Risk with the following discussion.

14.7 Classification variance decomposition

Amit, Blanchard, and Wilder (2000) provide a rigorous connection between the strength of individual classifiers and their correlation. To formalize this, we need to establish some notation. Consider a K -class classification problem where:

- X denotes the input feature vector
- $Y \in \{1, 2, \dots, K\}$ denotes the true class label
- $c \in \{1, 2, \dots, K\}$ represents a specific true class
- $d \in \{1, 2, \dots, K\}$ represents a candidate class label (the class we're voting for)

In an ensemble setting, let $h(X, d)$ denote an individual classifier (e.g., a single decision tree) that outputs a vote for class d given input X . Typically,

$h(X, d) \in \{0, 1\}$ for hard voting (1 if the classifier predicts class d , 0 otherwise) or $h(X, d) \in [0, 1]$ for soft voting (the probability that the classifier assigns to class d).

The ensemble aggregates votes from multiple classifiers. Let \mathbf{Q} denote the distribution over classifiers—this is the randomization scheme (e.g., bootstrap sampling, random feature selection) that generates the trees in the ensemble. The aggregate classifier $H_{\mathbf{Q}}(X, d)$ is the average vote for class d over all classifiers sampled from \mathbf{Q} :

$$H_{\mathbf{Q}}(X, d) = E_{\mathbf{Q}}[h(X, d)] = \frac{1}{B} \sum_{b=1}^B h_b(X, d)$$

where h_b are individual classifiers sampled from \mathbf{Q} and B is the number of classifiers in the ensemble. In practice, this is simply the proportion of trees that vote for class d (or the average probability assigned to class d across all trees).

The final classification decision $C_{\mathbf{Q}}(X)$ is the class that receives the most votes (or highest average probability):

$$C_{\mathbf{Q}}(X) = \arg \max_{d \in \{1, \dots, K\}} H_{\mathbf{Q}}(X, d)$$

Now, let P_c denote the population conditional probability distribution of a point X given $Y = c$, and let E_c and Var_c denote the associated conditional expectation and variance operators (i.e., $E_c[\cdot] = E[\cdot | Y = c]$ and $\text{Var}_c[\cdot] = \text{Var}[\cdot | Y = c]$). Define the vectors of average aggregates conditional on class c as

$$M_c(d) = E_c[H_{\mathbf{Q}}(X, d)] = E[H_{\mathbf{Q}}(X, d) | Y = c]$$

for $d = 1, \dots, K$. Intuitively, $M_c(d)$ represents the **expected vote share** for class d when the true class is c . Ideally, $M_c(c)$ (votes for the correct class) should be much larger than $M_c(d)$ (votes for incorrect classes).

The average conditional margin (ACM) for class c is defined as

$$\theta_c = \min_{d \neq c} (M_c(c) - M_c(d))$$

This θ_c measures the “safety gap” between the correct class and the closest competing class. A larger margin means the classifier is more robust.

We assume that $\theta_c > 0$. This assumption is very weak since it involves only the average over the population of class c . It is quite natural since one would not expect good classification results when it is violated. Indeed as shown below it is satisfied in all cases.

Given that $\theta_c > 0$, the error rate for class c depends on the extent to which the aggregate classifier $H_{\mathbf{Q}}(X, d)$ is concentrated around $M_c(d)$ for each $d = 1, \dots, K$. The simplest measure of concentration is the variance of $H_{\mathbf{Q}}(X, d)$ with respect to the distribution P_c . Using Chebyshev's inequality we write a coarse bound on the misclassification probability with respect to P_c as follows.

$$\begin{aligned} P_c(C_{\mathbf{Q}}(X) \neq c) &\leq P_c(H_{\mathbf{Q}}(X, c) < M_c(c) - \theta_c/2) \\ &\quad + \sum_{d \neq c} P_c(H_{\mathbf{Q}}(X, d) > M_c(d) + \theta_c/2) \\ &\leq \sum_{d=1}^K P_c(|H_{\mathbf{Q}}(X, d) - M_c(d)| > \theta_c/2) \\ &\leq \frac{4}{\theta_c^2} \sum_{d=1}^K \text{Var}_c[H_{\mathbf{Q}}(X, d)]. \end{aligned} \tag{10}$$

Of course Chebyshev's inequality is coarse and will not give very sharp results in itself, but we state it here as a landmark pointing to the relative importance of margin and variance, and to the tradeoff between the two quantities. To reduce error, we must either **increase the margin** θ_c (build stronger classifiers) or **decrease the variance** (add more diverse trees).

We rewrite each of the variance terms of the last equation as

$$\begin{aligned} \text{Var}_c[H_{\mathbf{Q}}(X, d)] &= E_c[H_{\mathbf{Q}}(X, d)]^2 - [E_c H_{\mathbf{Q}}(X, d)]^2 \\ &= E_{\mathbf{Q} \otimes \mathbf{Q}} E_c[h_1(X, d) h_2(X, d)] - E_{\mathbf{Q} \otimes \mathbf{Q}}[E_c[h_1(X, d)] E_c[h_2(X, d)]] \\ &= E_{\mathbf{Q} \otimes \mathbf{Q}} \text{Cov}_c[h_1(X, d), h_2(X, d)] \doteq \gamma_{c,d} \end{aligned} \tag{11}$$

Here, $E_{\mathbf{Q} \otimes \mathbf{Q}}$ denotes the expectation over the product measure of two independent draws from \mathbf{Q} . That is, h_1 and h_2 are two classifiers sampled independently from the distribution \mathbf{Q} , and $E_{\mathbf{Q} \otimes \mathbf{Q}}$ averages over all such pairs. This allows us to express the variance of the ensemble in terms of the covariance between pairs of individual classifiers.

This equation formalizes the intuition: to minimize error, we must minimize the covariance $\gamma_{c,d}$ between trees. This is precisely what **Random Forests** do by randomizing splits (Amit, Blanchard, and Wilder (2000)).

14.7.1 The Nearest Neighbor Insight

Why do trees work so well on complex data? T. Cover and Hart (1967) provided a fundamental result for nearest-neighbor classifiers: as sample size

grows, the error rate of a simple 1-Nearest Neighbor classifier is bounded by **twice the Bayes Risk** (the irreducible error).

$$R^* \leq R_{NN} \leq 2R^*(1 - R^*)$$

This means that a simple “look at your neighbor” strategy captures at least half the available signal. Decision trees can be viewed as **adaptive nearest-neighbor** models. Instead of using a fixed distance metric (which fails in high dimensions due to sparsity), trees learn a custom metric—dividing space into “blocky” regions based only on relevant features.



15

Forecasting

In 1766, Daniel Bernoulli developed the first mathematical model of disease transmission for smallpox, writing: “*I wish simply that, in matters which so closely concern the well being of the human race, no decision shall be made without all knowledge which a little analysis and calculation can provide.*” This pioneering work laid the foundation for epidemic modeling that continues to inform public health decisions today.

One of the most famous early applications involved the London Plague of 1665-1666, particularly in the village of Eyam near Sheffield. The data, shown below, tracked susceptible individuals (S) and infected individuals (I) over time:

Date (1666)	Susceptibles	Infectives
Initial	254	7
July 3	235	15
July 19	201	22
Aug 3	153	29
Aug 19	121	21
Sept 3	108	8
Sept 19	97	8
Oct 3	-	-
Oct 19	83	0

The initial population was $N = 261$, and by the end of the outbreak, 83 individuals remained susceptible (never infected). This simple dataset illustrates a fundamental challenge in time series forecasting: understanding the *mechanism* driving the dynamics.

The minister of the local church in Eyam, John Ashton, kept detailed records of the outbreak. The missing record on Oct 3 is due to the fact that John Ashton himself was infected and died on that day. Crucially, Ashton recognized the necessity of isolation; he strictly enforced a quarantine on his own household until his death. He also meticulously recorded daily infection counts, pioneering data collection practices nearly two centuries before Florence Nightingale.

Time series data appear across business, science, healthcare, and engineering. Unlike regression datasets, time series observations are not exchangeable—their order carries information. The modeling replacement for i.i.d. is typically conditional independence or state-space structure, but Bayesian updating logic remains the same as in Chapter 3.

This chapter describes the `bsts` software package, which fits sophisticated time series models with just a few lines of R code. The BSTS (Bayesian Structural Time Series) section is adapted from Steven Scott's blog post: "[Fitting Bayesian structural time series with the bststs R package](#)".

15.0.1 Epidemic Forecasting: A Motivating Example

The Eyam example highlights several themes central to this chapter:

1. *Self-reinforcing dynamics*: Disease spread accelerates with more infectants, creating nonlinear feedback loops
2. *State-space structure*: The compartments (S, E, I, R) represent latent states that evolve according to known dynamics
3. *Parameter uncertainty*: Even with detailed data, parameters like β and γ must be estimated
4. *Retrospective vs. prospective*: The Eyam analysis was performed in hindsight with complete data; real-time forecasting during an outbreak is considerably more challenging

Modern epidemic forecasting combines SEIR-type models with Bayesian methods to quantify uncertainty. While epidemic models differ from the business and economic time series we'll focus on in this chapter, they share the fundamental state-space structure that makes Bayesian structural time series models so powerful.

15.0.2 The `bsts` Package

The `bsts` package for R fits Bayesian structural time series models (also called state space models, Kalman filter models, or dynamic linear models). Available on CRAN via `install.packages("bsts")`, it is similar to Prophet but more customizable.

The chapter covers three examples: *Nowcasting* (local linear trend, seasonal models, spike-and-slab priors), *Long-term forecasting* (semilocal linear trend), and *Recession modeling* (non-Gaussian responses, serial dependence).

15.1 Structural time series models

A structural time series model is defined by two equations. The observation equation relates the observed data y_t to a vector of latent variables α_t known as the “state.”

$$y_t = Z_t^T \alpha_t + \epsilon_t.$$

The transition equation describes how the latent state evolves through time.

$$\alpha_{t+1} = T_t \alpha_t + R_t \eta_t.$$

The error terms ϵ_t and η_t are Gaussian and independent of everything else. The arrays Z_t , T_t and R_t are structural parameters. They may contain parameters in the statistical sense, but often they simply contain strategically placed 0’s and 1’s indicating which bits of α_t are relevant for a particular computation. An example will hopefully make things clearer.

The simplest useful model is the “local level model,” in which the vector α_t is just a scalar μ_t . The local level model is a random walk observed in noise.

$$\begin{aligned} y_t &= \mu_t + \epsilon_t \\ \mu_{t+1} &= \mu_t + \eta_t. \end{aligned}$$

Here $\alpha_t = \mu_t$, and Z_t , T_t , and R_t all collapse to the scalar value 1. Similar to Bayesian hierarchical models for nested data, the local level model is a compromise between two extremes. The compromise is determined by variances of $\epsilon_t \sim N(0, \sigma^2)$ and $\eta_t \sim N(0, \tau^2)$. If $\tau^2 = 0$ then μ_t is a constant, so the data are IID Gaussian noise. In that case the best estimator of y_{t+1} is the mean of y_1, \dots, y_t . Conversely, if $\sigma^2 = 0$ then the data follow a random walk, in which case the best estimator of y_{t+1} is y_t . Notice that in one case the estimator depends on all past data (weighted equally) while in the other it depends only on the most recent data point, giving past data zero weight. If both variances are positive then the optimal estimator of y_{t+1} winds up being “exponential smoothing,” where past data are forgotten at an exponential rate determined by the ratio of the two variances. Also notice that while the state in this model is Markov (i.e. it only depends on the previous state), the dependence among the observed data extends to the beginning of the series.

In the example above, one of the plots shows the price of Apple stock from 2021-01-01 to 2022-12-31. The other plot is a sequence generated from a random walk model fitted to the Apple price data. Can you spot which one is which?

Structural time series models are useful because they are flexible and modular. The analyst chooses the structure of α_t based on things like whether short-

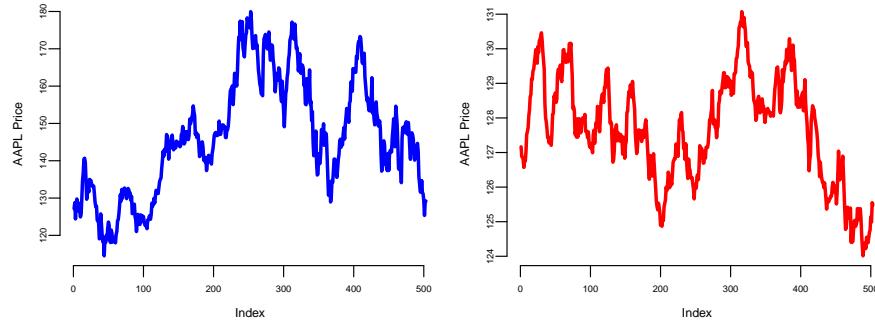


Figure 15.1: Apple Adjusted Closing Price Figure 15.2: Apple Adjusted Closing Price

or long-term predictions are more important, whether the data contains seasonal effects, and whether and how regressors are to be included. Many of these models are standard, and can be fit using a variety of tools, such as the `StructTS` function distributed with base R or one of several R packages for fitting these models (with the `dlm` package (Petris (2010), Campagnoli, Petrone, and Petris (2009)) deserving special mention). The `bsts` package handles all the standard cases, but it also includes several useful extensions, described in the next few sections through a series of examples. Each example includes a mathematical description of the model and example `bsts` code showing how to work with the model using the `bsts` software. To keep things short, details about prior assumptions are largely avoided.

Example 15.1 (Nowcasting). S. Scott and Varian (2014) and Steven L. Scott and Varian (2015) used structural time series models to show how Google search data can be used to improve short-term forecasts (“nowcasts”) of economic time series. The figure below shows the motivating data set from S. Scott and Varian (2014), which is also included with the `bsts` package. The data consist of the weekly initial claims for unemployment insurance in the US, as reported by the US Federal Reserve. Like many official statistics, they are released with delay and subject to revision. At the end of the week, the economic activity determining these numbers has taken place, but the official numbers are not published until several days later. For economic decisions based on these and similar numbers, it would help to have an early forecast of the current week’s number as of the close of the week. Thus the output of this analysis is truly a “nowcast” of data that has already happened rather than a “forecast” of data that will happen in the future.

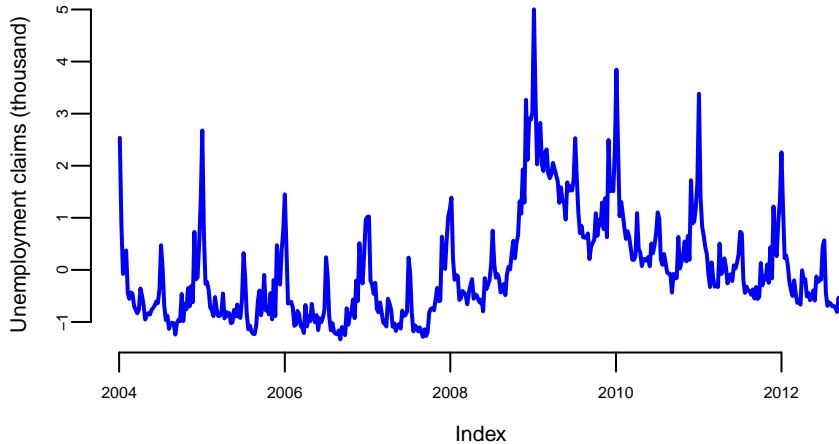


Figure 15.3: Weekly initial claims for unemployment in the US.

There are two sources of information about the current value y_t in the initial claims series: past values $y_{t-\tau}$ describing the time series behavior of the series, and contemporaneous predictors x_t from a data source which is correlated with y_t , but which is available without the delay exhibited by y_t . The time series structure shows an obvious trend (in which the financial and housing crises in 2008 - 2009 are apparent) as well as a strong annual seasonal pattern. The external data source explored by Scott and Varian was search data from Google trends with search queries such as “how to file for unemployment” having obvious relevance.

Scott and Varian modeled the data using a structural time series with three state components:

- trend μ_t
- seasonal pattern τ_t
- regression component $\beta^T x_t$.

The model is

$$\begin{aligned}
 y_t &= \mu_t + \tau_t + \beta^T x_t + \epsilon_t \\
 \mu_{t+1} &= \mu_t + \delta_t + \eta_{0t} \\
 \delta_{t+1} &= \delta_t + \eta_{1t} \\
 \tau_{t+1} &= - \sum_{s=1}^{S-1} \tau_t + \eta_{2t}.
 \end{aligned}$$

The trend component looks similar to the local level model above, but it has an extra term δ_t . Notice that δ_t is the amount of extra μ you can expect as $t \rightarrow t + 1$, so it can be interpreted as the slope of the local linear trend. Slopes normally multiply some x variable, but in this case $x = \Delta t$, which is omitted from the equation because it is always 1. The slope evolves according to a random walk, which makes the trend an integrated random walk with an extra drift term. The local linear trend is a better model than the local level model if you think the time series is trending in a particular direction and you want future forecasts to reflect a continued increase (or decrease) seen in recent observations. Whereas the local level model bases forecasts around the average value of recent observations, the local linear trend model adds in recent upward or downward slopes as well. As with most statistical models, the extra flexibility comes at the price of extra volatility.

The best way to understand the seasonal component τ_t is in terms of a regression with seasonal dummy variables. Suppose you had quarterly data, so that $S = 4$. You might include the annual seasonal cycle using 3 dummy variables, with one left out as a baseline. Alternatively, you could include all four dummy variables but constrain their coefficients to sum to zero. The seasonal state model takes the latter approach, but the constraint is that the S most recent seasonal effects must sum to zero in expectation. This allows the seasonal pattern to slowly evolve. Scott and Varian described the annual cycle in the weekly initial claims data using a seasonal state component with $S = 52$. Of course weeks don't neatly divide years, but given the small number of years for which Google data are available the occasional one-period seasonal discontinuity was deemed unimportant.

Let's ignore the regression component for now and fit a `bsts` model with just the trend and seasonal components.

The first thing to do when fitting a `bsts` model is to specify the contents of the latent state vector α_t . The `bsts` package offers a library of state models, which are included by adding them to a state specification (which is just a list with a particular format). The call to `AddLocalLinearTrend` above adds a local linear trend state component to an empty state specification (the `list()` in its first argument). The call to `AddSeasonal` adds a seasonal state component with 52 seasons to the state specification created on the previous line. The state vector α_t is formed by concatenating the state from each state model. Similarly, the vector Z_t is formed by concatenating the Z vectors from the two state models, while the matrices T_t and R_t are combined in block-diagonal fashion.

The state specification is passed as an argument to `bsts`, along with the data and the desired number of MCMC iterations. The model is fit using an MCMC algorithm, which in this example takes about 20 seconds to produce 1000 MCMC iterations. The returned object is a list (with class attribute `bsts`). You can see its contents by typing

```
## "sigma.obs"                  "sigma.trend.level"
## "sigma.trend.slope"          "sigma.seasonal.52"
## "final.state"                "state.contributions"
## "one.step.prediction.errors" "log.likelihood"
## "has.regression"             "state.specification"
## "prior"                      "timestamp.info"
## "model.options"              "family"
## "niter"                      "original.series"
```

The first few elements contain the MCMC draws of the model parameters. Most of the other elements are data structures needed by various S3 methods (`plot`, `print`, `predict`, etc.) that can be used with the returned object. MCMC output is stored in vectors (for scalar parameters) or arrays (for vector or matrix parameters) where the first index in the array corresponds to MCMC iteration number, and the remaining indices correspond to dimension of the deviate being drawn.

Most users won't need to look inside the returned `bsts` object because standard tasks like plotting and prediction are available through familiar S3 methods. For example, there are several plot methods available.

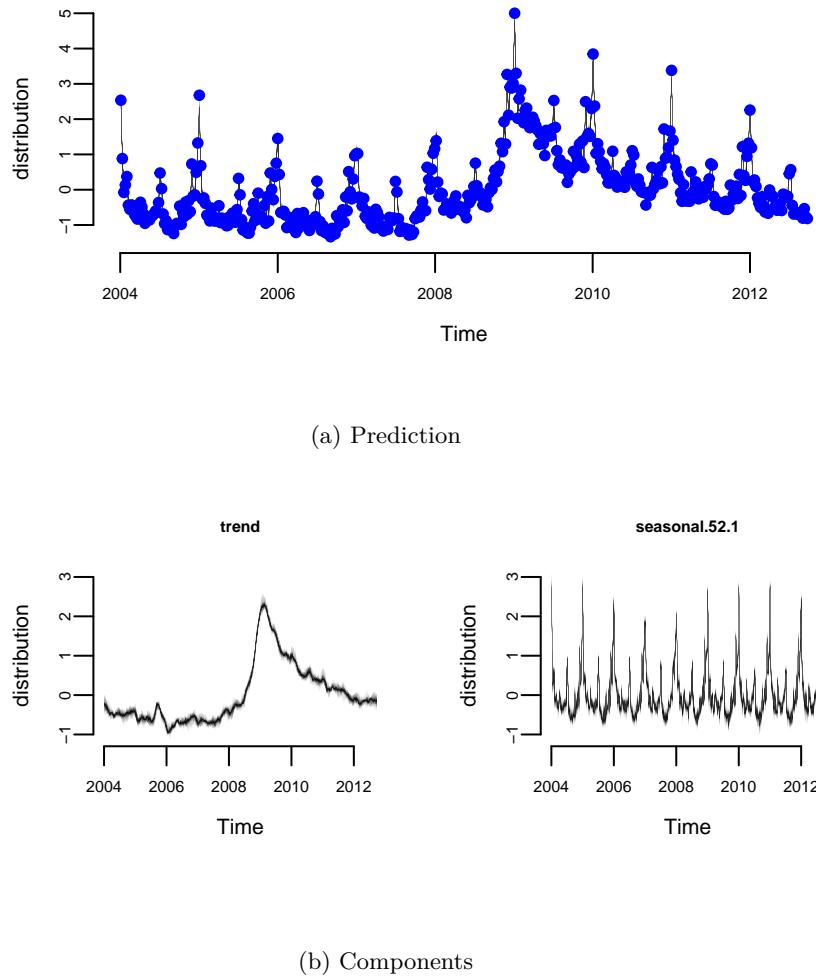


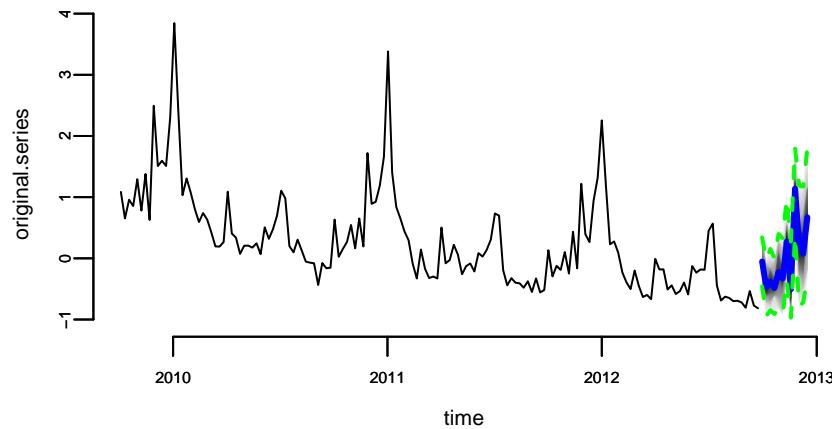
Figure 15.4: Structural time series model for unemployment claims

The Figure 15.4a above shows the posterior distribution of model state. Blue circles are actual data points. The Figure 15.4b shows the individual state components. The plot looks fuzzy because it is showing the marginal posterior distribution at each time point.

The default plot method plots the posterior distribution of the conditional mean $Z_t^T \alpha_t$ given the full data $y = y_1, \dots, y_T$. Other plot methods can be accessed by passing a string to the plot function. For example, to see the contributions of the individual state components, pass the string “components”

as a second argument, as shown above. The figure below shows the output of these two plotting functions. You can get a list of all available plots by passing the string `help` as the second argument.

To predict future values there is a `predict` method. For example, to predict the next 12 time points you would use the following commands.



The output of `predict` is an object of class `bsts.prediction`, which has its own `plot` method. The `plot.original = 156` argument says to plot the prediction along with the last 156 time points (3 years) of the original series.

15.2 Regression with spike and slab priors

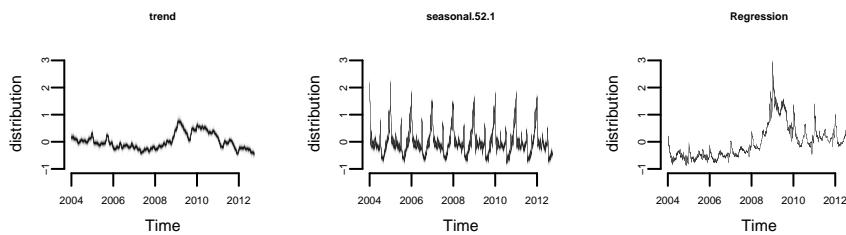
Now let's add a regression component to the model described above, so that we can use Google search data to improve the forecast. The `bsts` package only includes 10 search terms with the initial claims data set, to keep the package size small, but S. Scott and Varian (2014) considered examples with several hundred predictor variables. When faced with large numbers of potential predictors it is important to have a prior distribution that induces sparsity. A spike and slab prior is a natural way to express a prior belief that most of the regression coefficients are exactly zero.

A spike and slab prior is a prior on a set of regression coefficients that assigns each coefficient a positive probability of being zero. Upon observing data, Bayes' theorem updates the inclusion probability of each coefficient. When

sampling from the posterior distribution of a regression model under a spike and slab prior, many of the simulated regression coefficients will be exactly zero. This is unlike the “lasso” prior (the Laplace, or double-exponential distribution), which yields MAP estimates at zero but where posterior simulations will be all nonzero. You can read about the mathematical details of spike and slab priors in S. Scott and Varian (2014).

When fitting **bsts** models that contain a regression component, extra arguments captured by ... are passed to the SpikeSlabPrior function from the BoomSpikeSlab package. This allows the analyst to adjust the default prior settings for the regression component from the **bsts** function call. To include a regression component in a **bsts** model, simply pass a model formula as the first argument.

To examine the output you can use the same plotting functions as before. For example, to see the contribution of each state component you can type



It produces the contribution of each state component to the initial claims data, assuming a regression component with default prior. Compare to the previous model. The regression component is explaining a substantial amount of variation in the initial claims series.

There are also plotting functions that you can use to visualize the regression coefficients. The following commands plot posterior inclusion probabilities for predictors in the “initial claims” nowcasting example assuming an expected model size of 1 and 5.

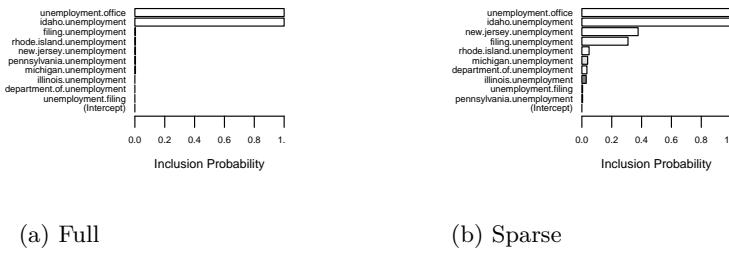


Figure 15.5: Variable Importance

The search term “unemployment office” shows up with high probability in both models. Increasing the expected model size from 1 (the default) to 5 allows other variables into the model, though “Idaho unemployment” is the only one that shows up with high probability.

Those probabilities are calculated from the histogram of the samples of each β calculated by the estimation algorithm (MCMC)

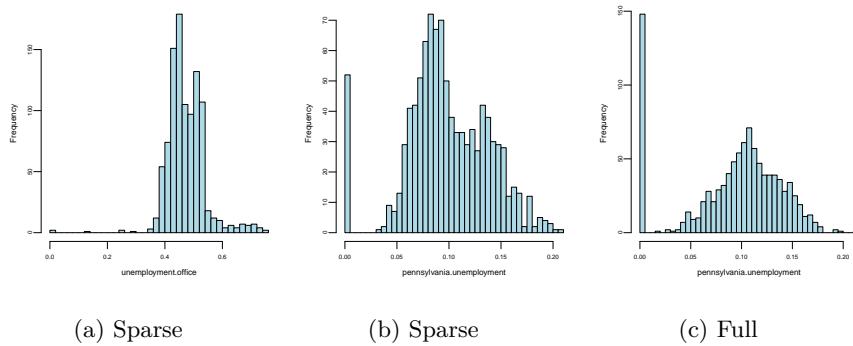


Figure 15.6: Sample from the distribution over two beta parameters

15.3 Model diagnostics: did the Google data help?

As part of the model fitting process, the algorithm generates the one-step-ahead prediction errors $y_t - E(y_t|Y_{t-1}, \theta)$, where $Y_{t-1} = y_1, \dots, y_{t-1}$, and the vector of model parameters θ is fixed at its current value in the MCMC algorithm. The one-step-ahead prediction errors can be obtained from the `bsts` model by calling `bsts.prediction.errors(model1)`.

The one step prediction errors are a useful diagnostic for comparing several

`bsts` models that have been fit to the same data. They are used to implement the function `CompareBstsModels`, which is called as shown below.

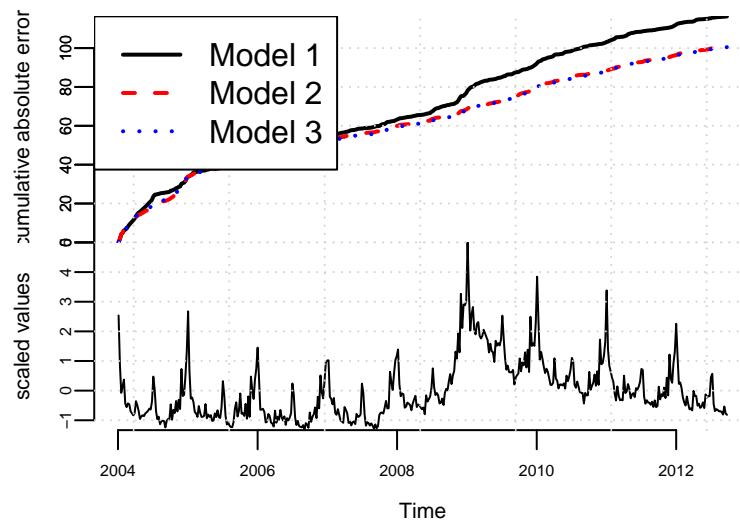


Figure 15.7: Comparison of Errors for the three models.

The bottom panel shows the original series. The top panel shows the cumulative total of the mean absolute one step prediction errors for each model. The final time point in the top plot is proportional to the mean absolute prediction error for each model, but plotting the errors as a cumulative total lets you see particular spots where each model encountered trouble, rather than just giving a single number describing each model's predictive accuracy. This figure shows that the Google data help explain the large spike near 2009, where model 1 accumulates errors at an accelerated rate, but models 2 and 3 continue accumulating errors at about the same rate they had been before. The fact that the lines for models 2 and 3 overlap in this figure means that the additional predictors allowed by the relaxed prior used to fit model 3 do not yield additional predictive accuracy.

Example 15.2 (Long-term forecasting). A common question about `bsts` is “which trend model should I use?” To answer that question it helps to know a

bit about the different models that the **bsts** software package provides, and what each model implies. In the local level model, the state evolves according to a random walk:

$$\mu_{t+1} = \mu_t + \eta_t.$$

If you place your eye at time 0 and ask what happens at time t , you find that $\mu_t \sim N(\mu_0, t\sigma_\eta^2)$. The variance continues to grow with t , all the way to $t = \infty$. The local linear trend is even more volatile. When forecasting far into the future, the flexibility provided by these models becomes a double-edged sword, as local flexibility in the near term translates into extreme variance in the long term.

An alternative is to replace the random walk with a stationary AR process. For example

$$\mu_{t+1} = \rho\mu_t + \eta_t,$$

with $\eta_t \sim N(0, \sigma_\eta^2)$ and $|\rho| < 1$. This model has stationary distribution

$$\mu_\infty \sim N \left(0, \frac{\sigma_\eta^2}{1 - \rho^2} \right),$$

which means that uncertainty grows to a finite asymptote, rather than infinity, in the distant future. The **bsts** package offers autoregressive state models through the functions **AddAr**, when you want to specify a certain number of lags, and **AddAutoAr** when you want the software to choose the important lags for you.

A hybrid model modifies the local linear trend model by replacing the random walk on the slope with a stationary AR(1) process, while keeping the random walk for the level of the process. The **bsts** package refers to this is the “semilocal linear trend” model.

$$\begin{aligned}\mu_{t+1} &= \mu_t + \delta_t + \eta_{0t} \\ \delta_{t+1} &= D + \rho(\delta_t - D) + \eta_{1t}\end{aligned}$$

The D parameter is the long-run slope of the trend component, to which δ_t will eventually revert. However, δ_t can have short-term autoregressive deviations from the long-term trend, with memory determined by ρ . Values of ρ close to 1 will lead to long deviations from D . To see the impact this can have on long-term forecasts, consider the time series of daily closing values for the S&P 500 stock market index over the last 5 years, shown below.



Figure 15.8: Daily closing values for the S&P 500 stock market index

Consider two forecasts of the daily values of this series for the next 360 days. The first assumes the local linear trend model. The second assumes the semilocal linear trend.

The figure below shows long-term forecasts of the S&P 500 closing values under the (left) local linear trend and (right) semilocal linear trend state models.

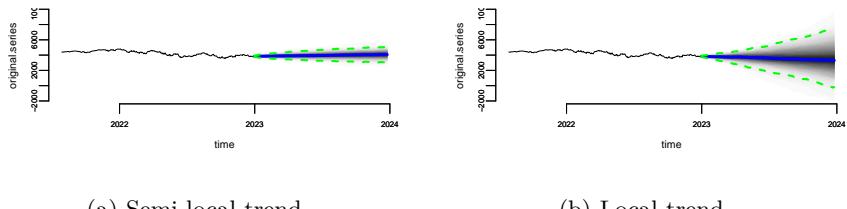


Figure 15.9: S&P 500 Prediction

Not only are the forecast expectations from the two models different, but the forecast errors from the local linear trend model are implausibly wide, including a small but nonzero probability that the S&P 500 index could close near zero in the next 360 days. The error bars from the semilocal linear trend model are far more plausible, and more closely match the uncertainty observed over the life of the series thus far.

Example 15.3 (Recession modeling using non-Gaussian data). Although we have largely skipped details about how the `bsts` software fits models, the Gaussian error assumptions in the observation and transition equations are important for the model fitting process. Part of that process involves running data through the Kalman filter, which assumes Gaussian errors in both the state and transition equations. In many settings where Gaussian errors are obviously inappropriate, such as for binary or small count data, one can introduce latent variables that give the model a conditionally Gaussian representation. Well known “data augmentation” methods exist for probit regression (Albert (1993)) and models with student-T errors (Rubin (2015)). Somewhat more complex methods exist for logistic regression (Frühwirth-Schnatter and Frühwirth (2007), Held and Holmes (2006), Gramacy and Polson (2012)) and Poisson regression (Frühwirth-Schnatter et al. (2008)). Additional methods exist for quantile regression (Benoit and Van den Poel (2012)), support vector machines (Nicholas G. Polson and Scott (2011)), and multinomial logit regression (Frühwirth-Schnatter and Frühwirth (2010)). These are not currently provided by the `bsts` package, but they might be added in the future.

To see how non-Gaussian errors can be useful, consider the analysis done by Berge, Sinha, and Smolyansky (2016), who used Bayesian model averaging (BMA) to investigate which of several economic indicators would best predict the presence or absence of a recession. We will focus on their nowcasting example, which models the probability of a recession at the same time point as the predictor variables. Berge, Sinha, and Smolyansky (2016) also analyzed the data with the predictors at several lags.

The model used in Berge, Sinha, and Smolyansky (2016) was a probit regression, with Bayesian model averaging used to determine which predictors should be included. The response variable was the presence or absence of a recession (as determined by NBER).

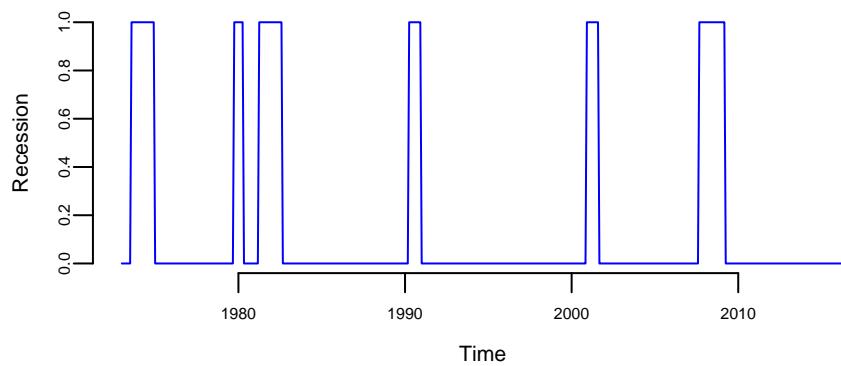


Figure 15.10: Recession periods identified by NBER

The BMA done by Berge, Sinha, and Smolyansky (2016) is essentially the same as fitting a logistic regression under a spike-and-slab prior with the prior inclusion probability of each predictor set to 1/2. That analysis can be run using the BoomSpikeSlab R package (Steven L. Scott (2022)), which is similar to `bsts`, but with only a regression component and no time series.

The logistic regression model is highly predictive, but it ignores serial dependence in the data. To capture serial dependence, consider the following dynamic logistic regression model with a local level trend model.

$$\text{logit}(p_t) = \mu_t + \beta^T x_t \\ \mu_{t+1} = \mu_t + \eta_t$$

Here p_t is the probability of a recession at time t , and x_t is the set of economic indicators used by Berge, Sinha, and Smolyansky (2016) in their analysis. The variables are listed in the table below

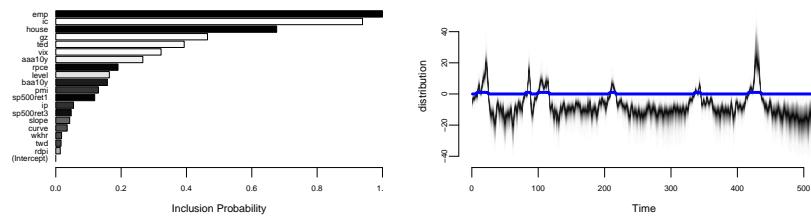
Variable	Definition/notes	Transformation
Financial Variables		
Slope of yield curve	10-year Treasury less 3-month yield	
Curvature of yield curve	2 x 2-year minus 3-month and 10-year	
GZ index	Gilchrist and Zakajsek (AER, 2012)	
TED spread	3-month ED less 3-month Treasury yield	
BBB corporate spread	BBB less 10-year Treasury yield	
S 500, 1-month return		1-month log diff.
S 500, 3-month return		3-month log diff.
Trade-weighted dollar		3-month log diff.
VIX	CBOE and extended following Bloom	
Macroeconomic Indicators		
Real personal consumption expend.		3-month log diff.
Real disposable personal income		3-month log diff.
Industrial production		3-month log diff.

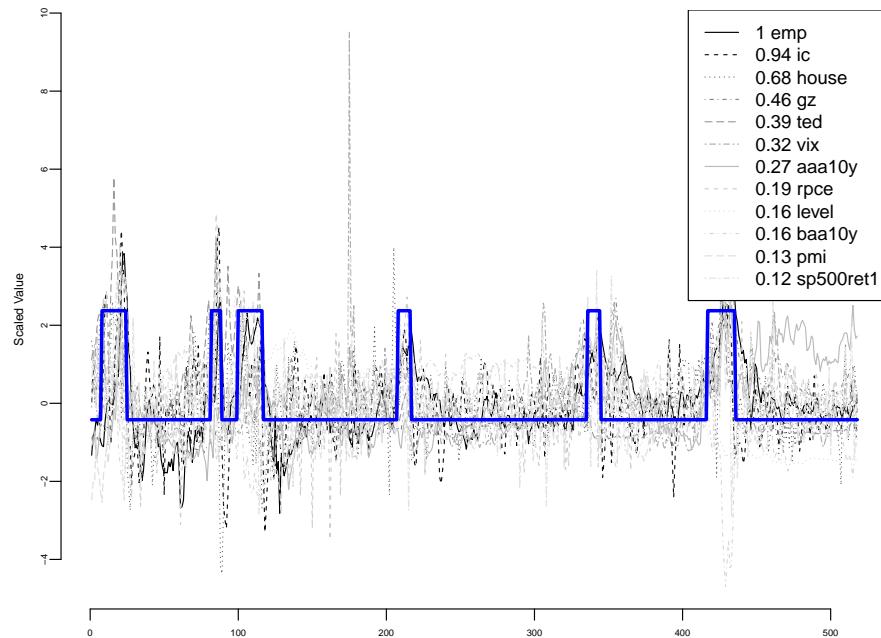
Variable	Definition/notes	Transformation
Housing permits		3-month log diff.
Nonfarm payroll employment		3-month log diff.
Initial claims	4-week moving average	3-month log diff.
Weekly hours, manufacturing		3-month log diff.
Purchasing managers index		3-month log dif

First, we prepare the data by shifting it by h , which is the forecast horizon.

To fit this model, we can issue the commands shown below.

Now let's plot the results





Notice that the distribution of p_t is moving to very large values during a recession, and to very small values outside of a recession. This effect captures the strong serial dependence in the recession data. Recessions are rare, but once they occur they tend to persist. Assuming independent time points is therefore unrealistic, and it substantially overstates the amount of information available to identify logistic regression coefficients.

15.4 Final Remarks on Structural Models

The preceding examples have shown that the `bsts` software package can handle several nonstandard, but useful, time series applications. These include the ability to handle large numbers of contemporaneous predictors with spike and slab priors, the presence of trend models suitable for long term forecasting, and the ability to handle non-Gaussian data. Further, `bsts` offers support for multiple seasonalities. For example, if you have several weeks of hourly data then you will have an hour-of-day effect as well as a day-of-week effect. You can model these using a single seasonal effect with 168 seasons (which would allow for different hourly effects on weekends and weekdays), or you can assume additive seasonal patterns using the `season.duration` argument to

`AddSeasonal.`

```
ss <- AddSeasonal(ss, y, nseasons = 24)
ss <- AddSeasonal(ss, y, nseasons = 7, season.duration = 24)
```

The latter specifies that each daily effect should remain constant for 24 hours. For modeling physical phenomena, `bsts` also offers trigonometric seasonal effects, which are sine and cosine waves with time varying coefficients. You obtain these by calling `AddTrig`. Time varying effects are available for arbitrary regressions with small numbers of predictor variables through a call to `AddDynamicRegression`.

In addition to the trend models discussed so far, the function `AddStudentLocalLinearTrend` gives a version of the local linear trend model that assumes student-t errors instead of Gaussian errors. This is a useful state model for short term predictions when the mean of the time series exhibits occasional dramatic jumps. Student-t errors can be introduced into the observation equation by passing the `family = "student"` argument to the `bsts` function call. Allowing for heavy tailed errors in the observation equation makes the model robust against individual outliers, while heavy tails in the state model provides robustness against sudden persistent shifts in level or slope. This can lead to tighter prediction limits than Gaussian models when modeling data that have been polluted by outliers. The observation equation can also be set to a Poisson model for small count data if desired.

Finally, the most recent update to `bsts` supports data with multiple observations at each time stamp. The Gaussian version of the model is

$$\begin{aligned} y_{it} &= \beta^T x_{it} + Z_t^T \alpha_t + \epsilon_{it} \\ \alpha_{t+1} &= T_t \alpha_t + R_t \eta_t, \end{aligned}$$

which is best understood as a regression model with a time varying intercept.

15.5 Beyond State-Space Models: Machine Learning Approaches

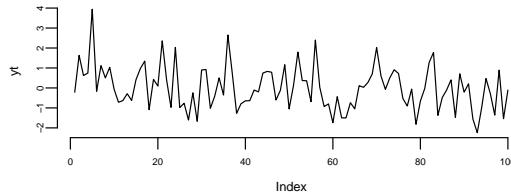
While structural time series models and `bsts` are powerful tools for interpretable probabilistic forecasting, the field of time series analysis has evolved rapidly, especially in the context of large-scale commercial applications. **However, a fundamental trade-off exists: State-space models offer transparency and uncertainty quantification, while modern machine learning approaches prioritize pure predictive power.**

15.5.1 Modern Era Forecasting

A recent post by the Amazon Science group Amazon (2021) describes the evolution of the time series algorithms used for forecasting from 2007 to 2021. Figure below shows the entire evolution of the algorithms.



They went from standard textbook time series forecasting methods to make predictions to the quantile-based transformer models. The main problem of the traditional TS models is that they assume stationarity. A stationary time series is one whose properties do not depend on the time at which the series is observed. For example, a white noise series is stationary - it does not matter when you observe it, it should look much the same at any point in time.



In other words, all the coefficients of a time series model do not change over time. We know how to deal with trends and seasonality quite well. Thus, those types of non-stationary are not an issue. Below some of the example of time series data. Although most of those are not stationary, we can model them using traditional techniques (Hyndman and Athanasopoulos (2021)).

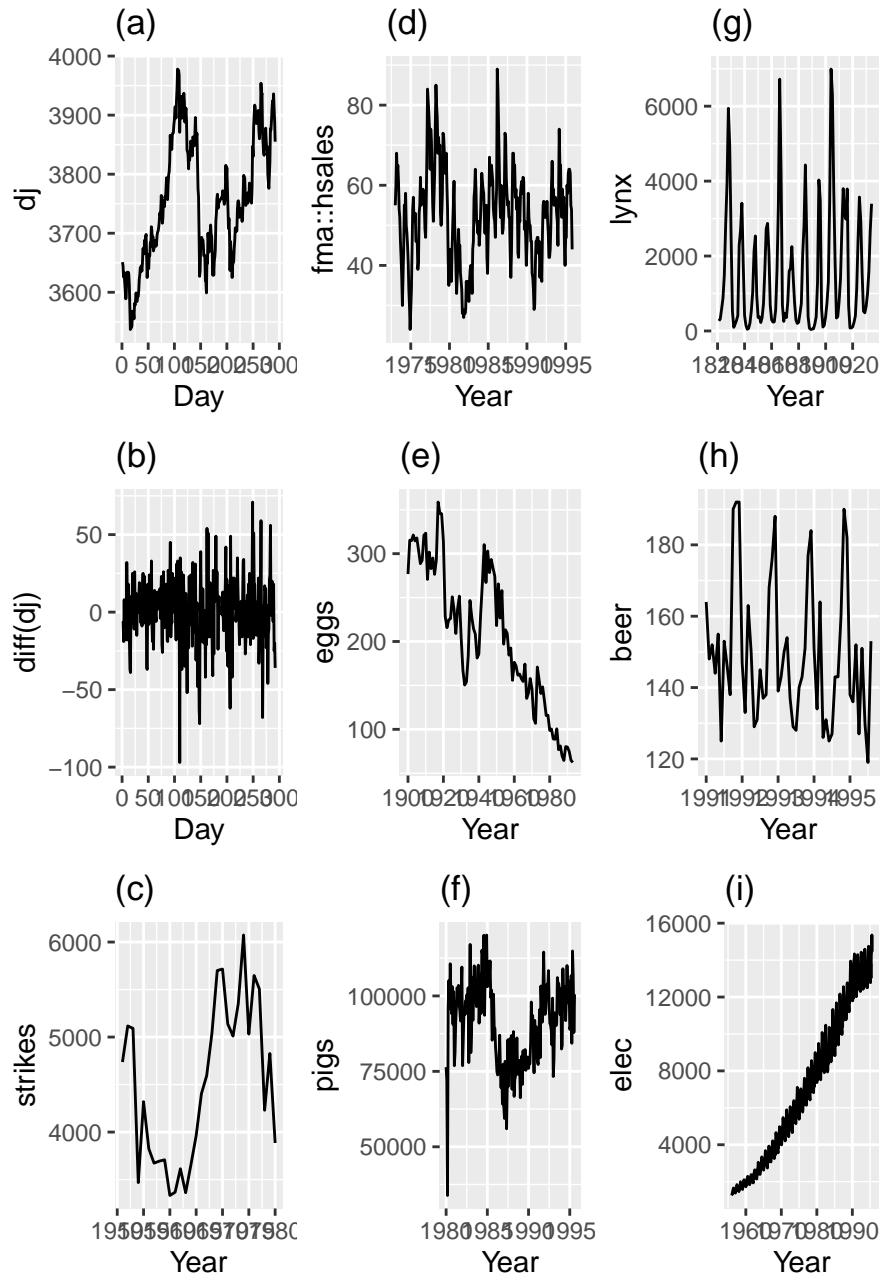


Figure 15.11: Which of these series are stationary? (a) Dow Jones index on 292 consecutive days; (b) Daily change in the Dow Jones index on 292 consecutive days; (c) Annual number of strikes in the US; (d) Monthly sales of new one-family houses sold in the US; (e) Annual price of a dozen eggs in the US (constant dollars); (f) Monthly total of pigs slaughtered in Victoria, Australia; (g) Annual total of lynx trapped in the McKenzie River district of northwest Canada; (h) Monthly Australian beer production; (i) Monthly Australian electricity production.

However, when you try to forecast for a time series with no prior history or non-recurrent “jumps”, like recessions, traditional models are unlikely to work well.

Amazon used a sequence of “patches” to hack the model and to make it produce useful results. All of those required manual feature engineering and led to less transparent and fragile models. One solution is to use random forests.

15.6 Quantile Regression Forests.

Most estimators during prediction return $E(Y|X)$, which can be interpreted as the answer to the question, what is the expected value of your output given the input?

Quantile methods, return y at q for which $F(Y = y|X) = q$ where q is the percentile and y is the quantile. One quick use-case where this is useful is when there are a number of outliers which can influence the conditional mean. It is sometimes important to obtain estimates at different percentiles, (when grading on a curve is done for instance.)

Note, Bayesian models return the entire distribution of $P(Y|X)$.

It is fairly straightforward to extend a standard decision tree to provide predictions at percentiles. When a decision tree is fit, the trick is to store not only the sufficient statistics of the target at the leaf node such as the mean and variance but also all the target values in the leaf node. At prediction, these are used to compute empirical quantile estimates.

The same approach can be extended to Random Forests. To estimate $F(Y = y|x) = q$ each target value in training ys is given a weight. Formally, the weight given to y_j while estimating the quantile is

$$\frac{1}{T} \sum_{t=1}^T \frac{\mathbb{1}(y_j \in L(x))}{\sum_{i=1}^N \mathbb{1}(y_i \in L(x))},$$

where $L(x)$ denotes the leaf that x falls into.

Informally, what it means that for a new unknown sample, we first find the leaf that it falls into at each tree. Then for each (X, y) in the training data, a weight is given to y at each tree in the following manner.

1. If it is in the same leaf as the new sample, then the weight is the fraction of samples in the same leaf.
2. If not, then the weight is zero.

These weights for each y are summed up across all trees and averaged. Now since we have an array of target values and an array of weights corresponding to these target values, we can use this to measure empirical quantile estimates.

Motivated by the success of gradient boosting models for predicting Walmart sales (kaggle (2020)), Januschowski et al. (2022) tries to explain why tree-based methods were so widely used for forecasting.

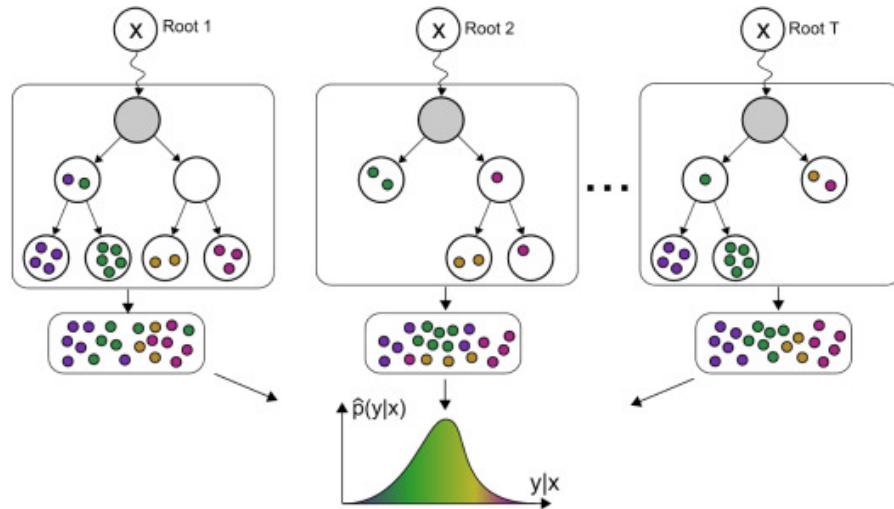


Figure 15.12: Januschowski et al. (2022)

15.7 Deep Learning for Time Series: Temporal Fusion Transformers

While Random Forests and Gradient Boosting Machines are powerful for tabular regression, deep learning architectures designed specifically for time series have gained significant traction. One state-of-the-art architecture is the **Temporal Fusion Transformer (TFT)** Lim et al. (2021).

TFT combines the strengths of several model components: 1. **Gating mechanisms** (GRN) to skip over unused components, allowing the model to adapt its depth. 2. **Variable selection networks** to select relevant input variables at each time step. 3. **Static covariate encoders** to integrate context (like location or store ID) that doesn't change over time. 4. **Temporal self-attention** (multi-head attention) to learn long-term dependencies across time steps. 5. **Quantile output** to generate probabilistic forecasts (intervals) rather than

just point estimates.

This architecture addresses a key critique of black-box models by offering **interpretability**: the attention weights can be visualized to show which past time points were most influential in making the current prediction.

15.7.1 Benchmark: BSTS vs. TFT on Initial Claims

To illustrate the trade-off between traditional probabilistic models and modern deep learning, we compare a **bsts** model against a TFT model on the **iclaims** dataset.

BSTS Implementation: We fit a standard local linear trend plus seasonal model using **bsts**.

```
# R: Fit BSTS
ss <- AddLocalLinearTrend(list(), y_train)
ss <- AddSeasonal(ss, y_train, nseasons = 52)
model1 <- bst(y_train, state.specification = ss, niter = 1000)
pred <- predict(model1, horizon = length(y_test))
```

TFT Implementation and The Reality of Tuning: We use the **pytorch-forecasting** library to train a **Temporal Fusion Transformer**. While Deep Learning models are often praised for their “auto-regressive” capabilities, achieving a sensible forecast on this dataset required a significant, iterative tuning process that highlights the practical challenges of these methods compared to **bsts**.

1. **Iterative Tuning:** Initially, the model produced flat lines or massive biases. Correcting this required:
 - **Feature Engineering:** Explicitly adding a “week of year” feature to help the model latch onto seasonality.
 - **Normalization Battles:** We had to revert from trend-aware scalers to standard scaling (**GroupNormalizer**) to ensure the model didn’t learn artificial offsets.
 - **Translation Invariance:** The most critical step was *removing* the global time index (**time_idx**) from the inputs. Without this, the model memorized the global trend of the training set and extrapolated it erroneously into the future.
2. **Ensembling:** To reduce the inherent variance of the neural network weights, we had to train an ensemble of 3 independent models and average their predictions.

Despite this “tedious” process, the TFT demonstrated remarkable **flexibility**. Unlike `bsts`, where we explicitly defined “Local Linear Trend” and “Seasonal” components, the TFT eventually learned these patterns entirely from the data structure, without us specifying the mathematical form of the trend.

```
# Python: Fit TFT (Ensemble of 3 models)
# Note: Global time index is excluded to enforce translation
    ↵ invariance
tft = TemporalFusionTransformer.from_dataset(
    training_dataset,
    learning_rate=0.015,
    hidden_size=64,
    attention_head_size=4,
    time_varying_known_reals=[], # No global time index
    loss=QuantileLoss([0.05, 0.5, 0.95])
)
trainer.fit(tft, train_dataloaders=train_dataloader)
```

Comparison Code: The full code for this case study, including the R script for BSTS and the Python script for TFT, along with the data files, is available in the `case_studies/tft/` directory of the book’s repository.

Results Comparison: The figure below compares the out-of-sample forecasts.

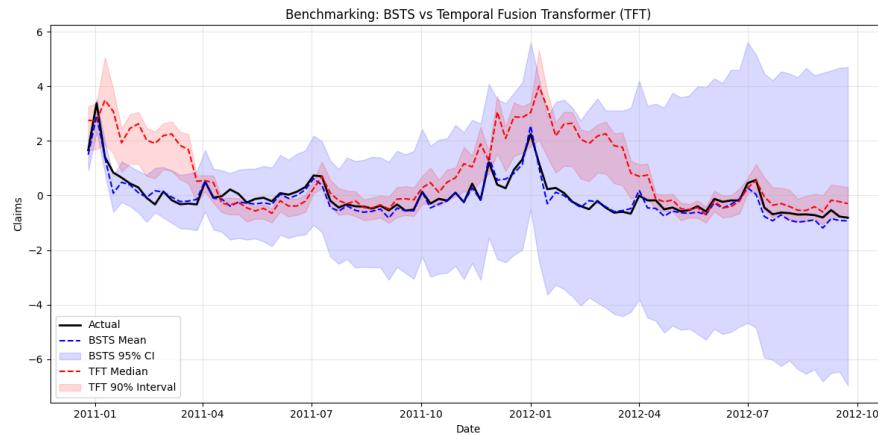


Figure 15.13: Benchmark: BSTS (Blue) vs. TFT (Red) Forecasts. Note the difference in Uncertainty Quantification.

Critical Assessment: Comparing the two forecasts reveals a fundamental difference in “honesty”:

- **BSTS (The Sensible Benchmark):** The BSTS forecast (Blue) is highly sensible. The predictive mean tracks the actual data closely throughout the horizon. More importantly, the **95% Credible Interval naturally expands** as the forecast moves further into the future. This is a desirable, “honest” feature of the structural model—it admits that uncertainty accumulates over time.
- **TFT (The Overconfident Learner):** The TFT forecast (Red), while capturing the rhythm of the seasonality, exhibits **undesirable overconfidence**. Its prediction intervals are surprisingly narrow and do not expand over time, implying it is equally confident about next week as it is about next year. Furthermore, for many periods, the actual observed data falls outside the TFT’s 90% interval, suggesting the model is underestimating the true noise in the system.

This comparison serves as a reminder: while Deep Learning offers immense flexibility and power for complex patterns, structural Bayesian models often provide more reliable, well-calibrated, and “sane” defaults for decision-making in standard business forecasting contexts.

15.8 Conclusion

This chapter focused on **Structural Time Series** models via `bsts` because they offer a practical balance for scientific and business applications:

1. **Interpretability:** BSTS decomposes time series into understandable components—trends, seasonality, and regressor effects—allowing analysts to explain *why* a forecast takes a particular value.
2. **Uncertainty Quantification:** The Bayesian framework provides credible intervals, not just point forecasts.
3. **Flexibility:** Priors like spike-and-slab enable robust variable selection even with noisy data or many predictors.

When pure predictive accuracy on massive datasets with complex non-linear interactions is the goal, gradient boosted trees and deep learning (Transformers) can outperform state-space models. Use `bsts` when you need mechanistic understanding and uncertainty quantification; consider deep learning when marginal accuracy gains justify the interpretability cost.

15.9 Advanced Theory: Under the Hood

[!TIP] **Optional Section:** This section delves into the mathematical machinery driving the models. Readers focused primarily on application can skip ahead to the Conclusion or the “Beyond State-Space Models” section.

[!NOTE] **A Note on Notation:** In this section, we transition to the standard engineering notation for state-space models to align with classic literature (like the Kalman Filter).

- **State:** x_t (previously α_t)
- **Transition Matrix:** F_t or A (previously T_t)
- **Observation Matrix:** G_t or H (previously Z_t^T)

This shift allows us to present the algorithms in their native mathematical form.

The models we have discussed in the `bsts` framework are all special cases of the **linear Gaussian state space model** (also known as the Dynamic Linear Model). While `bsts` provides a convenient high-level interface to fit these models using MCMC, understanding the underlying machinery provides deeper insight into how these forecasts are generated and updated as new data arrives.

The core algorithm for exact inference in linear Gaussian systems is the **Kalman Filter**. Originally developed for control engineering, it is the mathematical engine that computes the posterior distribution of the latent state α_t (or x_t in general notation) given the data up to time t . The recursive nature of the Kalman Filter—updating the state estimate with each new observation—is what allows structural time series models to adapt dynamically to changing trends and seasonal patterns.

The classic filtering and prediction algorithms for linear and Gaussian systems are described in Rudolph Emil Kalman (1960) and R. E. Kalman and Bucy (1961). Early work on discrete recursions for hidden Markov models are in Baum et al. (1970) who use an EM-type algorithm, Viterbi (1967) who provides a modal state filter estimate and recursions developed in Lindgren (1978). While these can be used to evaluate the marginal likelihood for the parameters they are computationally too intensive to solve the filtering and learning, Lindgren (1978). Steven L. Scott (2002) provides a review of FFBS algorithms for discrete HMMs.

Markov chain Monte Carlo (MCMC) algorithms for parameter learning in nonlinear non-Gaussian state space models were developed by Carlin, Polson,

and Stoffer (1992). For linear and Gaussian systems, Carter and Kohn (1994) introduced the filter forward and backwards sample (FFBS) algorithm, which efficiently draws the entire block of hidden states. For handling multinomial logit models, Steven L. Scott (2002) and Frühwirth-Schnatter et al. (2008) developed mixture of normals approximation methods. Additionally, West and Harrison (1997) proposed conditionally conjugate priors that allow parameters to be marginalized out of the updating equations, leading to more efficient inference procedures.

15.9.1 Kalman Filtering

The Normal/ Normal Bayesian learning model provides the basis for shrinkage estimation of multiple means and the basis of the Kalman filter for dynamically tracking a path of an object.

The Kalman filter is arguably the most common application of Bayesian inference. The Kalman filter assumes a linear and Gaussian state-space model:

$$y_t = x_t + \sigma \varepsilon_t^y \text{ and } x_t = x_{t-1} + \sigma_x \varepsilon_t^x,$$

where ε_t^y and ε_t^x are i.i.d. standard normal and σ and σ_x are known. The observation equation posits that the observed data, y_t , consists of the random-walk latent state, x_t , that is polluted by noise, $\sigma \varepsilon_t^y$. Further, σ_x/σ is the “signal-to-noise” ratio, measures the information content of the signal. As σ increases relatively to σ_x , the observations become noisier and less informative. The model is initialized via a prior distribution over x_0 , which simply for analytical tractability must be normally distributed, $x_0 \sim \mathcal{N}(\mu_0, \sigma_0^2)$.

The posterior distribution solves the filtering problem and is defined recursively via Bayes rule:

$$p(x_{t+1} | y^{t+1}) = \frac{p(y_{t+1} | x_{t+1}) p(x_{t+1} | y^t)}{p(y_{t+1} | y^t)} \propto p(y_{t+1} | x_{t+1}) p(x_{t+1} | y^t).$$

and the likelihood function, $p(y_{t+1} | x_{t+1})$. The predictive distribution summarizes all of the information about x_{t+1} based on lagged observations. The likelihood function summarizes the new information in y_{t+1} about x_{t+1} .

The Kalman filter relies on an inductive argument: assume that $p(x_t | y^t) \sim \mathcal{N}(\mu_t, \sigma_t^2)$ and then verify that $p(x_{t+1} | y^{t+1}) \sim \mathcal{N}(\mu_{t+1}, \sigma_{t+1}^2)$ with analytical expressions for the hyperparameters. To verify, note that since $p(x_t | y^t) \sim \mathcal{N}(\mu_t, \sigma_t^2)$, $x_t = \mu_t + \sigma_t \eta_t$ for some standard normal η_t . Substituting into the state evolution, the predictive is $x_{t+1} | y^t \sim \mathcal{N}(\mu_t, \sigma_t^2 + \sigma_x^2)$. Since

$p(y_{t+1} | x_{t+1}) \sim \mathcal{N}(x_{t+1}, \sigma^2)$, the posterior is

$$\begin{aligned} p(x_{t+1} | y^{t+1}) &\propto p(y_{t+1} | x_{t+1}) p(x_{t+1} | y^t) \propto \exp \left[-\frac{1}{2} \left(\frac{(y_{t+1} - x_{t+1})^2}{\sigma^2} + \frac{(x_{t+1} - \mu_t)^2}{\sigma_t^2 + \sigma_x^2} \right) \right] \\ &\propto \exp \left(-\frac{1}{2} \frac{(x_{t+1} - \mu_{t+1})^2}{\sigma_{t+1}^2} \right) \end{aligned}$$

where μ_{t+1} and σ_{t+1}^2 are computed by completing the square:

$$\frac{\mu_{t+1}}{\sigma_{t+1}^2} = \frac{y_{t+1}}{\sigma^2} + \frac{\mu_t}{\sigma_t^2 + \sigma_x^2} \text{ and } \frac{1}{\sigma_{t+1}^2} = \frac{1}{\sigma^2} + \frac{1}{\sigma_t^2 + \sigma_x^2}.$$

Here, inference on x_t is merely running the Kalman filter, that is, sequential computing μ_t and σ_t^2 , which are state sufficient statistics.

The Kalman filter provides an excellent example of the mechanics of Bayesian inference: given a prior and likelihood, compute the posterior distribution. The Kalman filter offers the most natural Bayesian approach for this setting. The same approach applied to learning fixed static parameters. In this case, $y_t = \mu + \sigma \varepsilon_t$, where $\mu \sim \mathcal{N}(\mu_0, \sigma_0^2)$ is the initial distribution. Using the same arguments as above, it is easy to show that $p(\mu | y^{t+1}) \sim \mathcal{N}(\mu_{t+1}, \sigma_{t+1}^2)$, where

$$\begin{aligned} \frac{\mu_{t+1}}{\sigma_{t+1}^2} &= \left(\frac{y_{t+1}}{\sigma^2} + \frac{\mu_t}{\sigma_t^2} \right) = \frac{(t+1)\bar{y}_{t+1}}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}, \\ \frac{1}{\sigma_{t+1}^2} &= \frac{1}{\sigma^2} + \frac{1}{\sigma_t^2} = \frac{(t+1)}{\sigma^2} + \frac{1}{\sigma_0^2}, \end{aligned}$$

and $\bar{y}_t = t^{-1} \sum_{i=1}^t y_i$.

This intuition extends directly to the state variable learning problem, offering a coherent framework for sequential learning. In this case, researchers often have different feelings about assuming a prior distribution over the state variable and a parameter. In the state filtering problem, it is difficult to separate the prior distribution and the likelihood. In fact, one could view the initial distribution over x_0 , the linear evolution for the state variable, and the Gaussian errors as the “prior” distribution.

Now consider linear multivariate Gaussian state space model:

$$\begin{aligned} y_t &= F_t x_t + \varepsilon_t \text{ where } \varepsilon_t \sim \mathcal{N}(0, \Sigma_t) \\ x_t &= G_t x_{t-1} + \varepsilon_t^x \text{ where } \varepsilon_t^x \sim \mathcal{N}(0, \Sigma_t^x) \end{aligned}$$

where we allow for heteroscedascity in the error variance-covariance matrices. We complete the model specification with a normal prior on the initial starting

condition $x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$. It is important to recognize that ε_t and ε_t^x need only be conditionally normal. There are a number of distributions of interest

$$\begin{aligned} \text{Filtering} : & p(x_t|y^t) \quad t = 1, \dots, T \\ \text{Forecasting} : & p(x_{t+1}|y^t) \quad t = 1, \dots, T \\ \text{Smoothing} : & p(x_t|y^{t+1}) \quad t = 1, \dots, T \\ \text{Prediction} : & p(y_{t+1}|y^t) \quad t = 1, \dots, T \end{aligned}$$

For known parameters with linearity and Gaussianity we have the following Kalman filter recursions for calculation these distributions.

The fundamental filtering relationship is based on the fact that the filtering distribution is of the form

$$p(x_t|y^t) \sim \mathcal{N}(\mu_{t|t}, \Sigma_{t|t}) \quad \text{and} \quad p(x_{t+1}|y^{t+1}) \sim \mathcal{N}(\mu_{t+1|t+1}, \Sigma_{t+1|t+1})$$

where $(\mu_{t+1|t+1}, \Sigma_{t+1|t+1})$ are related to $(\mu_{t|t}, \Sigma_{t|t})$ via the Kalman filter recursions. In the following, it is sometimes useful to write this as

$$p(x_t|y^t) \sim \mathcal{N}(\mu_{t|t}, \Sigma_{t|t}) \Rightarrow x_t = \mu_{t|t} + \Sigma_{t|t}^{\frac{1}{2}} \hat{\varepsilon}_t$$

where $\hat{\varepsilon}_t \sim \mathcal{N}(0, 1)$. Before we derive the filtering recursions and characterize the state filtering distribution we first find the forecasting distribution. The predictive or forecast distribution is defined as follows.

Predictive Distribution, $p(x_{t+1}|y^t)$.

The key distributions in Bayes rule are the predictive and the conditional state posterior given by

$$p(x_{t+1}|y^t) \sim \mathcal{N}(\mu_{t+1|t}, \Sigma_{t+1|t})$$

To compute the predictive or forecasting distribution note that:

$$p(x_{t+1}|y^t) = p(G_{t+1}x_t + \varepsilon_{t+1}^x|y^t) \sim \mathcal{N}(\mu_{t+1|t}, \Sigma_{t+1|t})$$

where the predictive moments are

$$\begin{aligned} \mu_{t+1|t} &= G_{t+1}\mu_t \\ \Sigma_{t+1|t} &= G_{t+1}\Sigma_t G_{t+1}^T + \Sigma_{t+1}^x. \end{aligned}$$

We now state and derive the main Kalman filtering recursions for linear Gaussian models with known parameters.

Filtering Distribution The classic Kalman filter characterisation of the state filtering distribution $p(x_{t+1}|y^{t+1})$ and moment recursions are given by

$$p(x_{t+1}|y^{t+1}) \sim \mathcal{N}(\mu_{t+1|t+1}, \Sigma_{t+1|t+1})$$

The updated posterior means and variances are defined by

$$\begin{aligned}\mu_{t+1|t+1} &= \mu_{t+1|t} + K_{t+1} e_{t+1} \\ \Sigma_{t+1|t+1} &= (I - K_{t+1} F_{t+1}) \Sigma_{t+1|t}\end{aligned}$$

where the Kalman gain K_{t+1} matrix and innovations vector e_{t+1} are

$$\begin{aligned}K_{t+1} &= \Sigma_{t+1|t} F_{t+1}^T (F_{t+1} \Sigma_{t+1|t} F_{t+1}^T + \Sigma_{t+1})^{-1} \\ e_{t+1} &= y_{t+1} - F_{t+1} \mu_{t+1|t}\end{aligned}$$

To prove this result we use the predictive distribution and an application of Bayes rule which implies that

$$\begin{aligned}p(x_{t+1}|y^{t+1}) &= p(x_{t+1}|y_{t+1}, y^t) \\ &= \frac{p(y_{t+1}|x_{t+1}) p(x_{t+1}|y^t)}{p(y_{t+1}|y^t)}.\end{aligned}$$

Under the normality assumption, the likelihood term is

$$p(y_{t+1}|x_{t+1}) = (2\pi)^{-\frac{p}{2}} |\Sigma_{t+1}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(y_{t+1} - F_{t+1} x_{t+1})^T \Sigma_{t+1}^{-1} (y_{t+1} - F_{t+1} x_{t+1})\right)$$

Combining with the exponent term from the state predictive distribution, then gives an exponent for the filtering distribution of the form

$$(y_{t+1} - F_{t+1} x_{t+1})^T \Sigma_{t+1}^{-1} (y_{t+1} - F_{t+1} x_{t+1}) + (x_{t+1} - \mu_{t+1|t})^T \Sigma_{t+1|t}^{-1} (x_{t+1} - \mu_{t+1|t})$$

Now we define the de-meanned state and innovations vectors,

$$\tilde{x}_{t+1} = x_{t+1} - \mu_{t+1|t} \text{ and } e_{t+1} = y_{t+1} - F_{t+1} \mu_{t+1|t}$$

Using the usual completing the square trick we can re-write the exponent as

$$(e_{t+1} - F_{t+1} \tilde{x}_{t+1})^T \Sigma_{t+1}^{-1} (e_{t+1} - F_{t+1} \tilde{x}_{t+1}) + \tilde{x}_{t+1}^T \Sigma_{t+1|t}^{-1} \tilde{x}_{t+1}$$

The sums of squares can be decomposed further as

$$\tilde{x}_{t+1}^T (F_{t+1}^T \Sigma_{t+1}^T + \Sigma_{t+1|t}^{-1}) \tilde{x}_{t+1} + 2\tilde{x}_{t+1}^T (F_{t+1}^T \Sigma_{t+1} e_{t+1}) + e_{t+1}^T \Sigma_{t+1}^{-1} e_{t+1}$$

The exponent is then a quadratic form implying that the vector \tilde{x}_{t+1}^T is normal distributed with the appropriate mean and variance-covariance matrix. The definitions are given by

$$\Sigma_{t+1|t+1} F_{t+1}^T \Sigma_{t+1} e_{t+1} \text{ and } \Sigma_{t+1|t+1} = (F_{t+1}^T \Sigma_{t+1}^{-1} F_{t+1} + \Sigma_{t+1|t}^{-1})^{-1}$$

respectively.¹ Hence, we obtain the identity

$$\Sigma_{t+1|t+1} = \left(F_{t+1}^T \Sigma_{t+1}^T F_{t+1} + \Sigma_{t+1|t}^{-1} \right)^{-1} = (I - K_{t+1} F_{t+1}) \Sigma_{t+1|t}$$

where $K_{t+1} = \Sigma_{t+1|t} F_{t+1}^T \left(F_{t+1} \Sigma_{t+1|t}^{-1} F_{t+1}^T + \Sigma_{t+1} \right)^{-1}$ is the Kalman gain matrix.

The mean of the $\tilde{x}_{t+1}^T = x_{t+1} - \mu_{t+1|t}$ distribution is then $K_{t+1} e_{t+1}$. Un demeaning the vector, we have $x_{t+1} = \tilde{x}_{t+1} + \mu_{t+1|t} = K_{t+1} e_{t+1}$ leads to the following distributional result

$$p(x_{t+1}|y^{t+1}) \sim \mathcal{N}(\mu_{t+1|t+1}, \Sigma_{t+1|t+1}),$$

The moments for the next filtering distribution are given by the classic recursions

$$\begin{aligned} \mu_{t+1|t+1} &= \mu_{t+1|t} + K_{t+1} (y_{t+1} - F_{t+1} \mu_{t+1|t}) \\ \Sigma_{t+1|t+1} &= (I - K_{t+1} F_{t+1}) \Sigma_{t+1|t}. \end{aligned}$$

There are two other distributions to compute: the data predictive $p(y_{t+1}|y^t)$ and the state smoothing distribution $p(x_t|y^{t+1})$. These are derived as follows.

The data predictive $p(y_{t+1}|y^t)$ is determined from the observation equation and the state predictive distribution as follows

$$\begin{aligned} y_{t+1} &= F_{t+1} x_{t+1} + \varepsilon_{t+1} \text{ with } \varepsilon_{t+1} \sim \mathcal{N}(0, \Sigma_{t+1}) \\ p(x_{t+1}|y^t) &\sim \mathcal{N}(\mu_{t+1|t}, \Sigma_{t+1|t}) \end{aligned}$$

Then substituting we have a predictive distribution for the next observation of the form

$$p(y_{t+1}|y^t) \sim \mathcal{N}\left(F_{t+1} \mu_{t+1|t}, F_{t+1} \Sigma_{t+1|t} F_{t+1}^T + \Sigma_{t+1}\right).$$

The state smoothing distribution $p(x_t|y^{t+1})$ is determined from the joint distribution, $p(x_t, x_{t+1}|y^t)$ as follows. First, factorise this joint distribution as

$$p(x_{t+1}, x_t|y^t) = p(x_{t+1}|x_t)p(x_t|y^t)$$

Then calculate the conditional posterior distribution, $p(x_{t+1}|x_t, y^{t+1})$ by Bayes rule as

$$p(x_{t+1}|x_t, y^{t+1}) = \frac{p(y_{t+1}|x_{t+1}) p(x_{t+1}|x_t) p(x_t|y^t)}{p(x_{t+1}|y^t)}$$

¹For computational purposes, we can further re-express these moments by using a matrix identity from Lindley and Smith (1972): for any matrices A_1, A_2, B_1, B_2 of appropriate dimensions and inverses we have the identity $(B_1 + A_1 B_2 A_1^T)^{-1} = (I - B_1^{-1} A_1 (A_1^T B_1^{-1} A_1 + B_2^{-1})^{-1} A_1^T) B_1^{-1}$. In our context of state filtering, we can apply this with $B_1 = \Sigma_{t+1|t}^{-1}$ and similarly defined other matrices to obtain the identities

Now, we can view the system as having two observations on x_{t+1} , namely

$$\begin{aligned}x_{t+1} &= F_{t+1}x_t + \Sigma_{t+1}^x \epsilon_{t+1}^x \\x_t &= \mu_{t|t} + \Sigma_{t|t}^{\frac{1}{2}} \hat{\epsilon}_t\end{aligned}$$

where the errors $\epsilon_{t+1}^x, \hat{\epsilon}_t$ are independent.

This leads to a joint posterior with an exponent that is proportional to

$$(x_{t+1} - G_{t+1}x_t)^T (\Sigma_{t+1}^x)^{-1} (x_{t+1} - G_{t+1}x_t) - (x_t - \mu_{t|t})^T \Sigma_{t|t}^{-1} (x_t - \mu_{t|t})$$

The first term comes from the state evolution and the second from the current filtering posterior. Completing the square gives

$$\begin{aligned}(x_{t+1} - G_{t+1}x_t)^T (\Sigma_{t+1}^x)^{-1} (x_{t+1} - G_{t+1}x_t) + (x_t - \mu_{t|t})^T \Sigma_{t|t}^{-1} (x_t - \mu_{t|t}) \\= (x_{t+1} - \mu_{t+1|t})^T \Sigma_{t|t}^{-1} (x_{t+1} - \mu_{t+1|t}) + (x_t - \mu_{t|t+1})^T \Sigma_{t|t+1}^{-1} (x_t - \mu_{t|t+1})\end{aligned}$$

which leads to the smoothed state moments

$$\begin{aligned}\mu_{t|t+1} &= \Sigma_{t|t+1} \left(\Sigma_{t|t} \mu_{t|t} + F_{t+1}^T (\Sigma_{t+1}^x)^{-1} x_{t+1} \right) \\ \Sigma_{t|t+1} &= F_{t+1}^T (\Sigma_{t+1}^x)^{-1} F_{t+1} + \Sigma_{t|t}^{-1}\end{aligned}$$

The Kalman filter recursions then follow by induction.

Example 15.4 (Kalman Filter for Robot Localization). The Kalman filter is a powerful tool for estimating the state of a system, given noisy observations. It is used in a wide range of applications, from tracking the position of a robot to estimating the state of a financial market. The Kalman filter is particularly useful when the state of the system is not directly observable, and must be inferred from noisy measurements.

Often KF is used for localization problem: given noisy measurements about the position of a robot and the motion model of the robot, the Kalman filter can estimate the true position of the robot. The Kalman filter is a recursive algorithm that estimates the state of a system at each time step, based on the state estimate from the previous time step and a new observation. We will use the language of state-space models in this example and will use the notation x_t to denote the state of the system at time t (parameter we are trying to estimate), and y_t to denote the observation at time t (observed data). The state-space model is given by

$$\begin{aligned}x_{t+1} &= Ax_t + w, \quad w \sim N(0, Q) \\y_t &= Gx_t + \nu, \quad \nu \sim N(0, R) \\x_0 &\sim N(\hat{x}_0, \Sigma_0),\end{aligned}$$

where A is the state transition matrix, G is the observation matrix, w is the process noise, and ν is the observation noise. The process noise and observation noise are assumed to be independent and normally distributed with zero mean and covariance matrices Q and R , respectively. The initial state x_0 is assumed to be normally distributed with mean \hat{x}_0 and covariance matrix Σ_0 . The Kalman filter provides a recursive algorithm for estimating the state of the system at each time step, based on the state estimate from the previous time step and a new observation. The state estimate is normal with mean \hat{x}_t and the covariance matrix Σ_t . The Kalman filter equations are given by

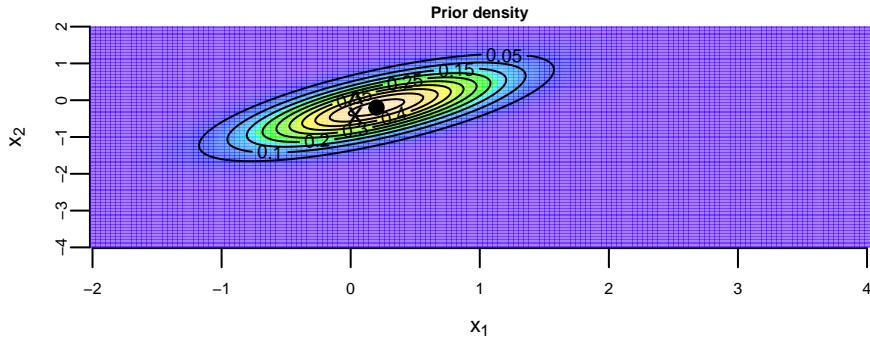
$$\begin{aligned}\hat{x}_{t+1} &= A\hat{x}_t + K_t(y_t - G\hat{x}_t) \\ K_t &= A\Sigma_t G^T (G\Sigma_t G^T + R)^{-1} \\ \Sigma_{t+1} &= A\Sigma_t A^T - K_t G \Sigma_t A^T + Q\end{aligned}$$

Kalman filter performs a multivariate normal-normal update using $N(A\hat{x}_t, A\Sigma_t A^T)$ as prior and $N(y_t, G\Sigma_t G^T + R)$ as likelihood. The posterior distribution is $N(\hat{x}_{t+1}, \Sigma_{t+1})$. Matrix K_t is called the Kalman gain and provides a weight on the residual between observed and prior $y_t - G\hat{x}_t$ in the update.

Assume our robot starts at $\hat{x}_0 = (0.2, -0.2)$ (x-y Cartesian coordinates) and initial covariance is

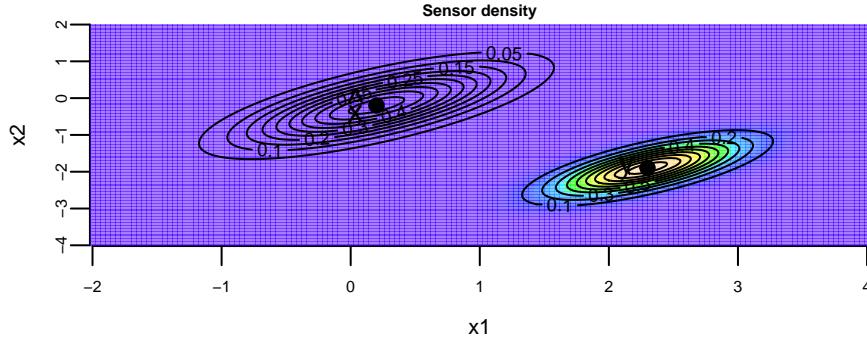
$$\Sigma_0 = \begin{bmatrix} 0.4 & 0.3 \\ 0.3 & 0.45 \end{bmatrix}.$$

The prior distribution of the robot's position can be visualized in R with a contour plot.



Now I get readings from GPS $y_0 = (2.3, -1.9)$ and I know from the manufacturer that the GPS has a covariance matrix of $R = 0.5\Sigma_0$. We assume the measurement matrix G to be identity matrix, thus

$$y_t = Gx_t + \nu_t = x_t + \nu, \quad \nu \sim N(0, R).$$



Now we combine our initial guess about the location x_0 with the measure noisy location data y_0 to obtain posterior distribution of the location of the robot $p(x | \hat{x}_0, \Sigma, R) = N(x | \hat{x}_f, \Sigma_f)$

$$\begin{aligned}\hat{x}_f &= (\Sigma^{-1} + R^{-1})^{-1}(\Sigma^{-1}\hat{x} + R^{-1}y) \\ \Sigma_f &= (\Sigma^{-1} + R^{-1})^{-1}\end{aligned}$$

Using the matrix inversion identity

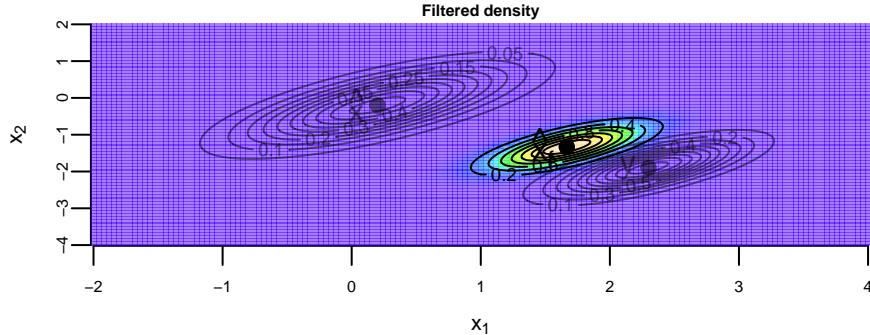
$$(A^{-1} + B^{-1})^{-1} = A - A(A + B)^{-1}A = A(A + B)^{-1}B$$

I can write the above as:

$$\begin{aligned}\hat{x}_f &= (\Sigma - \Sigma(\Sigma + R)^{-1}\Sigma)(\Sigma^{-1}\hat{x} + R^{-1}y) \\ &= \hat{x} - \Sigma(\Sigma + R)^{-1}\hat{x} + \Sigma R^{-1}y - \Sigma(\Sigma + R)^{-1}\Sigma R^{-1}y \\ &= \hat{x} + \Sigma(\Sigma + R)^{-1}(y - \hat{x}) \\ &= (1.667, -1.333) \\ \Sigma_f &= \Sigma - \Sigma(\Sigma + R)^{-1}\Sigma \\ &= \begin{bmatrix} 0.133 & 0.10 \\ 0.100 & 0.15 \end{bmatrix}\end{aligned}$$

In the more general case when G is not the identity matrix I have

$$\begin{aligned}\hat{x}_f &= \hat{x} + \Sigma G^T(G\Sigma G^T + R)^{-1}(y - G\hat{x}) \\ \Sigma_f &= \Sigma - \Sigma G^T(G\Sigma G^T + R)^{-1}G\Sigma\end{aligned}$$



Now I assume my robot moves according to the following model

$$x_t = Ax_{t-1} + w_t, \quad w_t \sim N(0, Q)$$

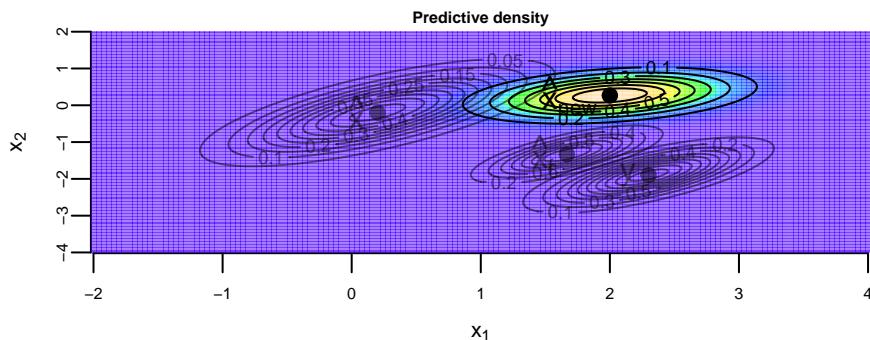
with

$$A = \begin{pmatrix} 1.2 & 0.0 \\ 0.0 & -0.2 \end{pmatrix}, \quad Q = 0.3\Sigma$$

Then the next location is normally distributed with the parameters

$$A = \begin{pmatrix} 1.2 & 0.0 \\ 0.0 & -0.2 \end{pmatrix}, \quad Q = 0.3\Sigma$$

Here $K = A\Sigma G^T (G\Sigma G^T + R)^{-1}$ is so-called Kalman gain matrix.



Forward filtering and Backwards Sampling

The Kalman filtering recursions lead to a fully recursive algorithm for characterizing $p(x|y)$ known as FFBS (Forward filtering and Backwards Sampling). This provides the counterpart to the Baum-Welch algorithm developed earlier

for HMMs. The details are as follows. The first step is to factorize the joint posterior distribution of the states via

$$\begin{aligned} p(x|y^T) &= p(x_T|y^T) \prod_{t=1}^{T-1} p(x_t|x_{t+1}, \dots, x_T, y^T) \\ &= p(x_T|y^T) \prod_{t=1}^{T-1} p(x_t|x_{t+1}, y^t) \end{aligned}$$

where we have used the fact that $p(x_t|x_{t+1}, \dots, x_T, y^T) = p(x_t|x_{t+1}, y^T)$ by the Markov property (conditional on x_{t+1} , x_t is independent of all x_{t+2} , etc.).

This forms the FFBS algorithm: forward-filtering, backward sampling algorithm for generating a block sample from $p(x|y^T)$. Filter forward using the Kalman recursions and obtain a sample from $p(x_T|y^T)$ and then backwards sample using $p(x_t|x_{t+1}, y^t)$ to generate a block draw of x . In what follows, we will often write

$$p(x|y^T) \sim FFBS$$

to denote that the FFBS algorithm can be used to generate a block draw.

Backwards Sampling.

The distribution of the final state given the data history $p(x_T|y^T)$ is given by the Kalman filter

$$p(x_T|y^T) \sim \mathcal{N}(\mu_T, \Sigma_T)$$

where (μ_T, Σ_T) are computed via the Kalman filter recursions. The second distribution comes from the factorization of $p(x_t, x_{t+1}|y^t)$ in the derivation of the Kalman filtering recursions. Hence, the conditional state filtering distribution given (x_{t+1}, y^t) is

$$p(x_t|x_{t+1}, y^t) \sim \mathcal{N}(\mu_{t|t+1}, \Sigma_{t|t+1})$$

where the one-step back smoothed moments are

$$\begin{aligned} \mu_{t|t+1} &= \Sigma_{t|t+1} \left(\Sigma_{t|t} \mu_{t|t} + F_{t+1}^T (\Sigma_{t+1}^x)^{-1} x_{t+1} \right) \\ \Sigma_{t|t+1} &= F_{t+1}^T (\Sigma_{t+1}^x)^{-1} F_{t+1} + \Sigma_{t|t}^{-1} \end{aligned}$$

as computed above. Then we can sequentially sample from this distribution.

15.9.2 HMM: Hidden Markov Models

The algorithms described in this section were originally developed by Baum et al. (1970) and Viterbi (1967). Baum developed original trading algorithms for Renaissance Technology which later became a multi-billion dollar hedge fund.

The algorithms are also known as the Baum-Welch and Viterbi algorithms, respectively. They are used to estimate the parameters of a Hidden Markov Model (HMM) from a sequence of observations. The HMM is a statistical model that describes a system that is assumed to be a Markov process with unobserved (hidden) states. These algorithms are now widely used in many applications, including speech recognition, bioinformatics, and finance.

Viterbi, on the other hand, was one of the founders of what is now known as Qualcomm, a multi-billion dollar semiconductor and telecommunications equipment company. Viterbi's algorithm is used to find the most likely sequence of hidden states in a Hidden Markov Model (HMM) given a sequence of observations.

Baum-Welch (1970) and Viterbi (1967) are the two famous discrete HMM algorithms.

Consider a model with a Hidden Chain or regime-switching variable

$$y_t = \mu(x_t) + \sigma(x_t) \varepsilon_t.$$

Suppose that x_t is a finite state Markov chain with a time-homogeneous transition matrix P with entries $\{p_{ij}\}$ which are given by

$$p_{ij} = P(x_t = i | x_{t-1} = j, \theta).$$

We define the marginal filtering and smoothing distributions

$$p_i^{t,t} = P(x_t = i | \theta, y^t) \text{ and } p_i^{t,T} = P(x_t = i | \theta, y^T)$$

and the corresponding joint filtering and smoothing matrices:

$$p_{ij}^{t,t} = P(x_{t-1} = i, x_t = j | \theta, y^t) \text{ and } p_{ij}^{t,T} = P(x_{t-1} = i, x_t = j | \theta, y^T).$$

The key to the algorithm is that we are just going to track the joint matrices, and then peel-off marginals from the rows and columns.

Forward-filtering

To derive the forward equations

$$\begin{aligned} p_{ij}^{t,t} &= P(x_{t-1} = i, x_t = j | \theta, y^t) \propto p(y_t, x_{t-1} = i, x_t = j | \theta, y^{t-1}) \\ &\propto p(y_t | x_t = j, \theta) p(x_t = j | x_{t-1} = i, \theta) p(x_{t-1} = i | \theta, y^{t-1}) \\ &\propto p(y_t | x_t = j, \theta) p_{ij}^{t-1,t-1}. \end{aligned}$$

This shows how to compute today's filtering distribution given the likelihood. The advantage of this is that it only requires matrix multiplication.

Backward-sampling

The result of the forward-filtering is the final observation $p_{ij}^{T,T}$. Like in the previous section, we can then filter in reverse to compute $p_{ij}^{t,T}$, which is required for the MCMC algorithm. We have that

$$\begin{aligned} p_{ij}^{t,T} &= p(x_{t-1} = i, x_t = j | \theta, y^T) \\ &\propto p(x_{t-1} = i | x_t = j, \theta, y^T) p(x_t = j | \theta, y^T) \\ &\propto p(x_{t-1} = i | x_t = j, \theta, y^t) p(x_t = j | \theta, y^T) \\ &\propto \frac{p(x_{t-1} = i, x_t = j | \theta, y^t)}{p(x_t = j | \theta, y^t)} p(x_t = j | \theta, y^T) \\ &\propto p_{ij}^{t,t} \frac{p_j^{t,T}}{p_j^{t,t}}. \end{aligned}$$

In deriving this, we have used the fact that

$$p(x_{t-1} = i | x_t = j, \theta, y^T) \propto p(x_{t-1} = i | x_t = j, \theta, y^t)$$

because conditional time t information, the past transition is independent of the future. This is the discrete-state version of the FFBS algorithm.

Smoothing: Forwards and Backwards

Let $y^T = \{y_1, \dots, y_T\}$ be a sequence of random variables where the conditional distribution

$$p(y^T | x^T) = p(y_1 | x_1) \prod_{t=2}^T p(y_t | x_t, y_{t-1})$$

where we suppress the dependence of the mixture components on a parameter θ . The full smoothing distribution can be written

$$p(x|y) = p(x_T | y^T) \prod_{t=1}^{T-1} p(x_t | x_{t+1}, \theta, y_{t+1})$$

Suppose $\{x_t\}$ follows a finite state Markov chain with initial distribution π_0 and transition probabilities

$$Q_t(r, s) = \Pr(x_t = s | x_{t-1} = r) \text{ and } P_t(t, r, s) = \Pr(x_{t-1} = r, x_t = s | y_1^t)$$

which we will compute sequentially. Let the current filtering distribution of the state be given by, for $t > 0$,

$$p_t(s) = \Pr(x_t = s | y^t) \text{ and } A_t(x_{t-1}, x_t, y_t) = p(x_{t-1}, x_t, y_t | y^{t-1})$$

By definition,

$$A_t(r, s, y_t) = \frac{p_{t-1}(r)Q_t(r, s)}{p(y_t | y_{t-1})}. \quad (15.1)$$

where the marginal likelihood is given by

$$p(y_t | y_1^{t-1}) = \sum_{r,s} A_t(r, s, y_t)$$

The filtered transition distribution

$$p_{trs} = A_t(r, s, y_t) / p(y_t | y^{t-1})$$

The forward-backward recursions are used to efficiently compute the observed data likelihood

$$p(y) = \sum_x p(y|x)p(x)$$

where $x = (x_1, \dots, x_T)$. We also need the posterior distribution $p(x|y)$ of the latent Markov chain given observed data. The recursions consist of a forward step that computes the distribution of the t 'th transition given all the data up to time t , and a backward recursion that updates each distribution to condition on all observed data.

The forward recursion operates on the set of transition distributions, represented by a sequence of matrices $P_t = (p_{trs})$. Computing

$$p_t(s) = \sum_r p_{trs}$$

sets up the next step in the recursion. The recursion is initialized by replacing p with p^0 in equation Equation 15.1.

The observed data log-likelihood can be computed as

$$\log p(y) = \log p(y_1) + \sum_{t=2}^T \log p(y_t|y^{t-1})$$

With appropriate use of logarithms, p and $p(y_t|y^{t-1})$ need only ever be evaluated on the log scale.

The stochastic version of the backward recursion simulates from

$$p(x|y).$$

Begin with the factorization

$$p(x|y) = p(x_T|y^T) \prod_{t=1}^{T-1} p(x_t|x_{t+1}^T, y).$$

Then notice that, given x_{t+1} , x_t is conditionally independent of y_{t+1}^T and all later x 's. Thus

$$p(x_t|x_{t+1}, y) = P(x_t = r|x_{t+1} = s, y^{t+1}) \propto p_{t+1rs}$$

Therefore, if one samples (x_{t-1}, x_t) from the discrete bivariate distribution given by P_t and then repeatedly samples x_t from a multinomial distribution proportional to column x_{t+1} of P_{t+1} then $x = (x_1, \dots, x_T)$ is a draw from $p(x|y)$.

15.9.3 Mixture Kalman filter

We can also introduce a λ_t state variable and consider a system

$$\begin{aligned} y_t &= F_{\lambda_t} x_t + D_{\lambda_t} \epsilon_t \\ x_t &= G_{\lambda_t} x_{t-1} + B_{\lambda_t} v_t \end{aligned}$$

The Kalman filter gives moments of the state filtering distribution

$$x_t|\lambda^t, y^t \sim \mathcal{N}(\mu_{t|t}, \Sigma_{t|t})$$

Here we assume that the iid auxiliary state variable shocks $\lambda_t \sim p(\lambda_t)$.

First we marginalize over the state variable x_t . Then we can track the sufficient statistics for the hidden z_t variable dynamically in time, namely $(z_{t|t}, S_t)$, just the Kalman filter moments, in the conditionally Gaussian and discrete cases Lindgren (1978). Then we re-sample $(z_{t|t}, S_t, \theta)^{(i)}$ particles.

15.9.4 Regime Switching Models

The general form of a continuous-time regime switching model is

$$dy_t = \mu(\theta, x_t, y_t) dt + \sigma(\theta, x_t, y_t) dB_t$$

where x_t takes values in a discrete space with transition matrix $P_{ij}(t)$ with parameters $\theta = (\theta_1, \dots, \theta_J)$. Common specifications assume the drift and diffusion coefficients are parametric functions and the parameters switch over time. In this case, it is common to write the model as

$$dy_t = \mu(\theta_{x_t}, y_t) dt + \sigma(\theta_{x_t}, y_t) dB_t.$$

Scott (2002) provides a fast MCMC algorithm for state filtering by adapting the FFBS algorithm. Time discretized the model is:

$$y_t = \mu(\theta_{x_t}, y_{t-1}) + \sigma(\theta_{x_t}, y_{t-1}) \varepsilon_t.$$

Note that we use the standard notation from discrete-time models where the time index on the Markov state is equal to the current observation. The discrete-time transition probabilities are

$$p_{ij} = P(x_t = i | x_{t-1} = j)$$

and we assume, a priori, that the transition functions are time and state invariant. The joint likelihood is given by

$$p(y|x, \theta) = \prod_{t=1}^T p(y_t | y_{t-1}, x_{t-1}, \theta)$$

where $p(y_t | y_{t-1}, x_{t-1}, \theta) = N(\mu(\theta_{x_{t-1}}, y_{t-1}), \sigma^2(\theta_{x_{t-1}}, y_{t-1}))$.

Clifford-Hammersley implies that the complete conditionals are given by $p(\theta|x, s, y)$, $p(s|x, \theta, y)$, and $p(x|s, \theta, y)$. Conditional on the states and the transition probabilities, updating the parameters is straightforward. Conditional on the states, the transition matrix has a Dirichlet distribution, and updating this is also straightforward. To update the states use FFBS.

An important component of regime switching models is the prior distribution. Regime switching models (and most mixture models) are not formally identified. For example, in all regime switching models, there is a labeling problem: there is no unique way to identify the states. A common approach to overcome this identification issue is to order the parameters.

Changepoint Problems

Smith (1975) introduced the single changepoint problem from a Bayesian perspective. Consider a sequence of random variables y_1, \dots, y_T which has a change-point at time τ in the sense that

$$y_t | \theta_k \sim \begin{cases} p(y|\theta_1) & \text{for } 1 \leq i \leq \tau \\ p(y|\theta_2) & \text{for } \tau + 1 \leq i \leq T \end{cases}$$

This can be rewritten as a state space model

$$y_t = \theta_{x_t} + \sigma_{x_t} \epsilon_t$$

where x_t has a Markov transition evolution.

An idea that appears to be under-exploited is that of “*model reparametrisation*”. The multiple change-point problem, which is computationally expensive if approached directly, has a natural model reparametrisation that makes the implementation of MCMC methods straightforward (see Chib (1998)). Specifically, suppose that the data generating process $y^T = \{y_1, \dots, y_T\}$ is given by a sequence of conditionals $f(y_t | y^{t-1}, \theta_k)$ for parameters θ_k that change at unknown change-points $\{\tau_1, \dots, \tau_k\}$.

The model parameterization is based on using a hidden Markov state space model with a vector of latent variables s_t where $s_t = k$ indicates that y_t is drawn from $p(y_t | y^{t-1}, \theta_k)$. Let the prior distribution on the s_t 's have transition matrix where $p_{ij} = P(s_t = j | s_{t-1} = i)$ is the probability of jumping regimes. With this model parameterization the k th change occurs at τ_k if $s_{\tau_k} = k$ and $s_{\tau_k+1} = k + 1$. The reparameterisation automatically enforces the order constraints on the change-points and is it very easy to perform MCMC analysis on the posterior distribution. This provides a more efficient strategy for posterior computation. MCMC analysis of the s_t 's is straightforward and the posterior for the τ_k 's can be obtained by inverting the definition above. Hence

$$p(\tau = t | y) = p(x_t = 1 | y)$$

The alternative is single state updating conditional on τ which is slow for finding the multiple-changepoints.

15.10 Particle Learning for General Mixture Models

Particle learning (PL) offers a powerful and flexible approach for sequential inference in general mixture models. Unlike traditional MCMC methods, which require repeated passes over the entire dataset and can be computationally demanding, particle learning operates in an online fashion. This means it can efficiently update inference as new data arrives, making it particularly well-suited for large or high-dimensional datasets and real-time applications.

The particle learning framework is designed to efficiently and sequentially learn from a broad class of mixture models. At its core, the approach models data as arising from a mixture distribution:

$$f(z) = \int k(z; \theta) dG(\theta)$$

where G is a discrete mixing measure and $k(z; \theta)$ is a kernel parameterized by θ . The generality of this formulation allows PL to be applied to a wide variety of models, including finite mixture models, Dirichlet process mixtures, Indian buffet processes, and probit stick-breaking models. This flexibility is a significant advantage, as it enables practitioners to use PL across many settings without needing to redesign the inference algorithm for each new model structure.

In addition to its generality, particle learning provides an alternative to MCMC for tasks such as online model fitting, marginal likelihood estimation, and posterior cluster allocation. Its sequential nature makes it particularly attractive for streaming data and scenarios where computational resources are limited.

A general mixture model can be described by three components: a likelihood, a transition equation for latent allocations, and a prior over parameters. Specifically,

- The likelihood is given by $p(y_{t+1}|k_{t+1}, \theta)$, representing the probability of the next observation given the current allocation and parameters.
- The transition equation $p(k_{t+1}|k^t, \theta)$ governs how the latent allocation variables evolve over time, where $k^t = \{k_1, \dots, k_t\}$ denotes the history of allocations.
- The parameter prior $p(\theta)$ encodes prior beliefs about the mixture component parameters.

This structure can be expressed in a state-space form:

$$y_{t+1} = f(k_{t+1}, \theta) \quad (15.2)$$

$$k_{t+1} = g(k^t, \theta) \quad (15.3)$$

where the first equation is the observation model and the second describes the evolution of the latent allocation states.

The mixture modeling framework described above is closely related to hidden Markov models (HMMs). In this context, the observed data y_t are assumed to be generated from a mixture, with allocation variables k_t determining which mixture component is responsible for each observation. The parameters θ_{k_t} for each component are drawn from the mixing measure G . This structure allows for both standard mixture models, where each observation is assigned to a single component, and more general latent feature models, where multivariate allocation variables k_t allow an observation to be associated with multiple components simultaneously.

A central concept in particle learning is the essential state vector \mathcal{Z}_t , which is tracked over time. This vector is constructed to be sufficient for sequential inference, meaning that it contains all the information needed to compute the posterior predictive distribution for new data, update the state as new observations arrive, and learn about the underlying parameters:

- Posterior predictive: $p(y_{t+1}|\mathcal{Z}_t)$
- Posterior updating: $p(\mathcal{Z}_{t+1}|\mathcal{Z}_t, y_{t+1})$
- Parameter learning: $p(\theta|\mathcal{Z}_{t+1})$

15.10.1 The Particle Learning Algorithm

Particle learning approximates the posterior distribution $p(\mathcal{Z}_t|y^t)$ with a set of equally weighted particles $\{\mathcal{Z}_t^{(i)}\}_{i=1}^N$. When a new observation y_{t+1} becomes available, the algorithm proceeds in two main steps:

1. **Resample:** The current set of particles is resampled with weights proportional to the predictive likelihood $p(y_{t+1}|\mathcal{Z}_t^{(i)})$. This step focuses computational effort on the most plausible states given the new data.
2. **Propagate:** Each resampled particle is then propagated forward by sampling from the transition distribution $p(\mathcal{Z}_{t+1}|\mathcal{Z}_t^{(i)}, y_{t+1})$, thus updating the state to incorporate the new observation.

This two-step process is grounded in Bayes' theorem, where the resampling step corresponds to updating the posterior with the new data, and the propagation step advances the state according to the model dynamics. After these

steps, the set of particles provides an updated approximation to the posterior $p(\mathcal{Z}_{t+1}|y^{t+1})$.

One important distinction between particle learning and standard particle filtering methods is that the essential state vector \mathcal{Z}_t does not necessarily need to include the full history of allocation variables k^t to be sufficient for inference. This makes PL both more efficient and more flexible than many existing particle filtering approaches for mixture models. Furthermore, the order of resampling and propagation steps is reversed compared to standard filters, which helps mitigate particle degeneracy and improves performance in mixture modeling contexts.

Particle learning also provides an efficient mechanism for sampling from the full posterior distribution of the allocation vector $p(k^t|y^t)$. This is achieved using a backwards uncertainty update, which allows for the recovery of smoothed samples of the allocation history. For each particle, and for each time step in reverse order, the allocation variable k_r is sampled with probability proportional to the product of the likelihood and the prior for that allocation, given the state vector. This results in an algorithm with computational complexity linear in the number of particles, making it practical even for large datasets.

The particle learning framework is applicable to a wide range of density estimation problems involving mixtures of the form

$$f(y; G) = \int k(y; \theta) dG(\theta)$$

There are many possible choices for the prior on the mixing measure G . Common examples include finite mixture models, which use a finite number of components; Dirichlet process mixtures, which allow for an infinite number of components via a stick-breaking construction; beta two-parameter processes; and kernel stick-breaking processes. Each of these priors offers different modeling flexibility and computational properties, and the choice depends on the specific application and desired level of model complexity.

In some cases, it is useful to consider a collapsed state-space model, where the predictive distribution for a new observation is expressed as an expectation over the mixing measure G given the current state vector:

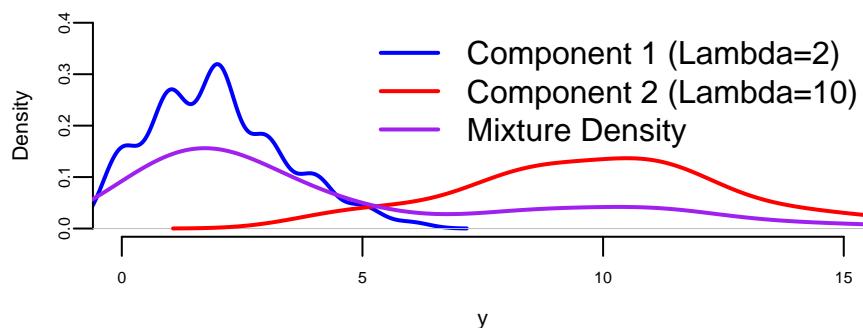
$$\mathbb{E}[f(y_{t+1}; G)|\mathcal{Z}_t] = \int k(y_{t+1}; \theta) d\mathbb{E}[G(\theta)|\mathcal{Z}_t]$$

If t observations have been allocated to m_t mixture components, the posterior expectation of G can be written as a weighted sum of the base measure and point masses at the component parameters. The predictive density then combines contributions from both new and existing components, weighted according to their posterior probabilities.

Particle learning offers a versatile and efficient framework for sequential inference in general mixture models. By representing the posterior with a set of particles and updating these particles as new data arrives, PL enables real-time model fitting, efficient posterior allocation, and flexible density estimation across a wide range of mixture modeling scenarios. Its ability to handle both finite and infinite mixture models, as well as latent feature models, makes it a valuable tool for modern statistical analysis.

Example 15.5 (Particle Learning for Poisson Mixture Models). We will implement Particle Learning (PL) for a finite mixture of Poisson distributions based on the example from Carlos M. Carvalho et al. (2010). This example follows Algorithm 1 for finite mixture models from Section 2.1 of the paper.

We generate data from a mixture of two Poisson distributions ($\lambda_1 = 2$ with weight 0.7, $\lambda_2=10$ with weight 0.3).

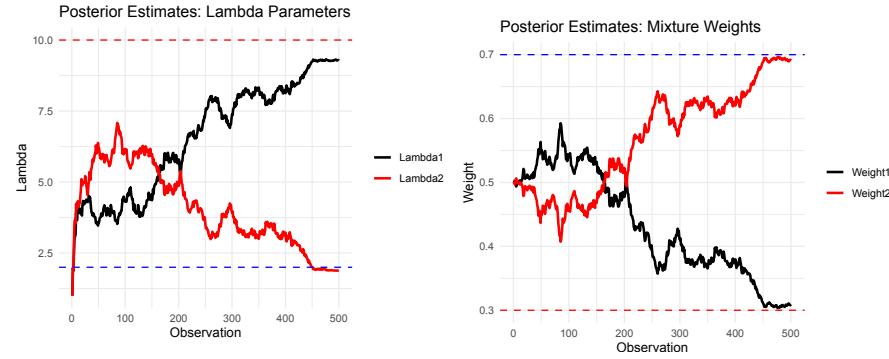


The code below implements the Particle Learning algorithm using the following steps: 1 Particle Initialization: - Each particle tracks sufficient statistics: - s : Sum of observations per component - n : Count of observations per component 2. PL Algorithm: - Resample: Particles are weighted by the posterior predictive probability of the next observation - Propagate: For each particle: - Compute component allocation probabilities - Sample component assignment - Update sufficient statistics - Learn: Store posterior estimates of λ parameters and mixture weights

The key features of this implementation are the use of posterior predictive with Poisson-Gamma conjugacy and allocation of probabilities by combining prior weights and likelihood.

Now we are ready to plot the results

The first plot shows the posterior estimates of λ parameters converging to true values (2 and 10). The second plot shows mixture weights converging to true weights (0.7 and 0.3). Convergence typically occurs after 100-200 observations. Particle degeneracy is mitigated through systematic resampling



(a) Posterior Estimates: Lambda Parameters

(a) Posterior Estimates: Mixture Weights

Advantages of PL for Mixtures:

1. Sequential Updating: Processes observations one-at-a-time
2. Efficiency: Only tracks sufficient statistics, not full history
3. Flexibility: Easily extends to other mixture types (DP, IBP, etc.)
4. Real-time Inference: Posterior updates after each observation

This implementation demonstrates PL's ability to handle finite mixtures, but the same framework extends to infinite mixtures (DP mixtures) and other general mixture models described in the paper by modifying the propagation and resampling steps.

16

Model Selection

“When you have eliminated the impossible, whatever remains, however improbable, must be the truth.” - Sherlock Holmes

Model selection is the art of choosing the model that best captures the true signal while avoiding the trap of fitting noise. Holmes’ words apply directly: when confronted with data, we must navigate countless possible models, each representing a different hypothesis, and systematically eliminate those that fail to generalize.

The next chapter (Chapter 17) covers regularization techniques—Ridge regression, LASSO, and their Bayesian interpretations—which can also be viewed as model selection mechanisms. Here we focus on the fundamental considerations: the bias-variance tradeoff, overfitting and underfitting, computational constraints, and whether our goal is prediction or interpretation. We draw on Breiman’s influential distinction between the “two cultures” of statistical modeling, then develop practical methodologies including cross-validation and information criteria.

16.1 Fundamental Considerations in Model Selection

16.1.1 Model Complexity and Generalization

Choosing the right model for the relationship between x and y involves navigating a fundamental trade-off between model complexity and generalization ability. If the chosen model is too simple (e.g., linear regression when the true relationship is polynomial), it might underfit the data and fail to capture important relationships, leading to high bias and poor performance on both training and test data. Conversely, a model that is excessively complex (e.g., high-degree polynomials or deep neural networks trained on insufficient data) risks overfitting by memorizing training examples rather than learning the underlying pattern. This results in excellent training performance but poor generalization to unseen examples. This challenge is exacerbated when

dealing with non-linear relationships, high-dimensional data, or noisy signals, where the optimal complexity is not immediately obvious. Systematic experimentation with different model architectures, regularization techniques, and hyperparameter tuning is often required to strike the right balance between signal capture and noise rejection.

16.1.2 Overfitting and Underfitting

Overfitting arises when a statistical learning algorithm captures stochastic noise rather than the underlying signal. This phenomenon typically manifests when a model possesses excessive degrees of freedom relative to the training data size, allowing it to “memorize” specific examples. While such a model may achieve near-zero error on the training set, it fails to generalize to new data because it has learned idiosyncrasies specific to the training sample rather than the general population structure. Common indicators of overfitting include a diverging gap between training and validation error curves or performance degradation on held-out data during training.

Underfitting, conversely, occurs when a model lacks the sufficient complexity to capture the true underlying structure between inputs and outputs. This often results from specifying a model class that is too rigid (e.g., fitting a linear model to quadratic data) or from over-regularization. An underfit model exhibits high bias, performing poorly on both training and test datasets. Unlike overfitting, underfitting is characterized by consistently high error rates across all data partitions, indicating a fundamental inability to model the signal.

16.1.3 Data Quality and Quantity

The reliability of predictive models is intrinsically linked to the quality and richness of the available data. Noisy, incomplete, or biased data inevitably leads to suboptimal model performance. While sufficient data volume is crucial for learning complex relationships, data *quality* often plays a more decisive role. Issues such as missing values, inconsistent formatting, label noise, and sampling bias are pervasive in real-world applications.

To address these challenges, the industry has seen the rise of data-centric AI platforms. Services like Scale AI and Toloka offer human-in-the-loop solutions for high-quality data annotation and validation. These platforms leverage globally distributed workforces to perform tasks ranging from image segmentation to text classification, ensuring that the ground truth labels used for training are accurate. By implementing rigorous quality control mechanisms—such as consensus voting among multiple annotators and dynamic skill-based routing—these services mitigate the risks associated with poor data quality.

16.1.4 Model Explainability

In many high-stakes domains—such as healthcare, finance, and criminal justice—predictive accuracy alone is insufficient. Stakeholders require *interpretability*: a clear understanding of how a model arrives at its decisions. This need drives the trade-off between using complex “black box” models (like deep neural networks) and simpler, transparent models (like logistic regression or decision trees).

Regulatory frameworks, including the EU’s GDPR, increasingly mandate a “right to explanation,” compelling organizations to deploy systems that are not just accurate but also accountable. While we explore specific techniques for achieving this—such as LIME, SHAP, and attention mechanisms—in detail later in this chapter, it is vital to recognize at the outset that the choice of model often dictates the ceiling of explainability.

16.1.5 Computational Cost

Training and serving predictive models can be computationally expensive, particularly for deep learning architectures operating on massive datasets. In resource-constrained environments, this necessitates a trade-off between model performance and computational efficiency.

Development of specialized hardware has played a pivotal role in addressing this. Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) allow for massive parallelization of matrix operations, reducing training times from weeks to hours. However, *inference* cost remains a challenge for deployment.

Edge computing—processing data locally on devices rather than in the cloud—has emerged as a solution for low-latency applications like autonomous driving and IoT. To enable this, techniques such as *quantization* (reducing numerical precision from 32-bit floats to 8-bit integers) and *model pruning* (removing redundant connections) are frequently employed. These methods allow complex models to run efficiently on mobile and embedded hardware with minimal loss of accuracy.

16.1.6 Ethical Considerations

Predictive models are not value-neutral; their deployment can have profound societal consequences. Ethical failures often manifest as *algorithmic bias*, where models perpetuate or amplify existing discrimination. For instance, facial recognition systems trained on imbalanced datasets have demonstrated significantly higher error rates for darker-skinned individuals. Similarly, hiring

algorithms trained on historical data may learn to replicate past discriminatory hiring practices.

Fairness in machine learning is an active area of research, dealing with metrics like statistical parity and equalized odds. However, maximizing fairness often requires trade-offs with predictive accuracy, necessitating careful ethical judgment during model development.

Privacy is another key concern. Deep learning models can inadvertently memorize sensitive training data, making them vulnerable to inversion attacks. *Differential privacy* offers a rigorous mathematical framework to mitigate this risk by adding calibrated noise to computations, ensuring that the model's output does not reveal whether any specific individual's data was included in the training set.

Finally, accountability is essential. “Algorithmic impact assessments” and “audits” are becoming standard practice to evaluate potential harms before deployment, ensuring that systems serve the public good while minimizing risk.

16.2 Prediction vs Interpretation

Predictive models can serve two distinct purposes: prediction and interpretation. These goals often conflict. Interpretation requires understanding the relationship between input and output variables, which typically demands simpler, more transparent models.

A model that excels at prediction might not be suitable for interpretation. For example, a complex deep neural network might achieve high predictive accuracy but provide little insight into how the input variables influence the output. Conversely, a simple linear model might be highly interpretable but lack the flexibility to capture complex relationships in the data. A key advantage of linear models is their ability to serve both purposes effectively, unlike more complex models with many parameters that can be difficult to interpret.

Interpretation problems typically require simpler models. We prioritize models that are easy to interpret and explain, even if they have slightly lower predictive accuracy. The evaluation metrics also differ: for interpretation, we typically use the coefficient of determination (R-squared) or p-values, which provide insights into the model's fit and the statistical significance of the estimated relationships.

The choice between using a model for prediction or interpretation depends on the specific task and desired outcome. If the primary goal is accurate predictions, a complex model with high predictive accuracy might be preferred,

even if it is less interpretable. However, if understanding the underlying relationships and causal mechanisms is crucial, a simpler and more interpretable model might be chosen, even if it has slightly lower predictive accuracy. Interpretive models are commonly used in scientific research, social sciences, and other fields where understanding the underlying causes and relationships is crucial.

In practice, it's often beneficial to consider both prediction and interpretation when building and evaluating models. However, it is not unusual to build two different models, one for prediction and one for interpretation. This allows for a more nuanced analysis of the data and can lead to better insights than using a single model for both purposes.

16.2.1 Breiman's Two Cultures

Let x be a high-dimensional input containing a large set of potentially relevant data, and let y represent an output (or response) to a task that we aim to solve based on the information in x . Breiman [2000] summarizes the difference between statistical and machine learning philosophy as follows:

“There are two cultures in the use of statistical modeling to reach conclusions from data. One assumes that the data are generated by a given stochastic data model. The other uses algorithmic models and treats the data mechanism as unknown.”

“The statistical community has been committed to the almost exclusive use of data models. This commitment has led to irrelevant theory, questionable conclusions, and has kept statisticians from working on a large range of interesting current problems.”

“Algorithmic modeling, both in theory and practice, has developed rapidly in fields outside statistics. It can be used both on large complex data sets and as a more accurate and informative alternative to data modeling on smaller data sets. If our goal as a field is to use data to solve problems, then we need to move away from exclusive dependence on data models and adopt a more diverse set of tools.”

Deep learning predictors offer several advantages over traditional predictors:

- Input data can include all data of possible relevance to the prediction problem at hand
- Nonlinearities and complex interactions among input data are accounted for seamlessly

- Overfitting is more easily avoided than with traditional high-dimensional procedures
- Fast, scalable computational frameworks (such as TensorFlow) are available

Tree-based models and deep learning models exemplify the “algorithmic culture” that Breiman describes. These models can capture complex, non-linear relationships in data without requiring explicit specification of the functional form. However, this flexibility comes at the cost of interpretability. While decision trees offer some interpretability through their hierarchical structure, deep neural networks are often considered “black boxes” due to their complex, multi-layered architecture.

The trade-off between interpretability and accuracy is a central theme in modern machine learning. Simple models like linear regression are highly interpretable but may lack the flexibility to capture complex patterns. Complex models like deep neural networks can achieve high accuracy but are difficult to interpret. This has led to the development of various techniques for making complex models more interpretable, including feature importance measures, attention mechanisms, and surrogate models that approximate the behavior of complex models with simpler, more interpretable ones.

16.3 Diagnostics for Model Assumptions

What makes a good model? If the goal is prediction, then the model is as good as its predictions. The easiest way to visualize the quality of predictions is to plot y versus \hat{y} . Most of the time we use empirical assessment of model quality. However, sometimes theoretical bounds can be derived for a model that describe its accuracy. For example, in the case of the linear regression model, the prediction interval is defined by

$$\hat{y} \pm s \sqrt{1 + \frac{1}{n} + \frac{(x - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2}}$$

where s is the standard deviation of the residuals. The prediction interval is the confidence interval for the prediction. The prediction interval is wider than the confidence interval because it includes the uncertainty in the prediction.

Assume we have a predictive model

$$y = f(x) + \epsilon$$

and we have some modeling assumption regarding the distribution of ϵ . For example, when we use linear regression or BART, we assume that ϵ follows a

normal distribution. One simple approach to test if observed samples $\epsilon_1, \dots, \epsilon_n$ follow a specific distribution is to use *Exploratory Data Analysis (EDA)*.

The most common tools for exploratory data analysis are Q-Q plots, scatter plots, and bar plots/histograms.

A Q-Q plot compares the quantiles of your data with the quantiles of a theoretical distribution (like normal, exponential, etc.). A quantile is the fraction (or percent) of points below the given value. That is, the i -th quantile is the point x for which $i\%$ of the data lies below x . On a Q-Q plot, if the two datasets come from a population with the same distribution, we should see the points forming a line that's roughly straight. More precisely, if the two datasets x and y come from the same distribution, then the points $(x_{(i)}, y_{(i)})$ should lie roughly on the line $y = x$. If y comes from a distribution that's linear in x , then the points $(x_{(i)}, y_{(i)})$ should lie roughly on a line, but not necessarily on the line $y = x$.

Example 16.1 (Normal Q-Q plot). Figure 16.1 shows the normal Q-Q plot for the Data on birth weights of babies born in a Brisbane hospital on December 18, 1997. The data set contains 44 records. A more detailed description of the data set can be found in [UsingR manual](#).

```
babyboom <- read.csv("../data/babyboom.csv")
qqnorm(babyboom$wt, bg = "lightblue")
qqline(babyboom$wt, col = "red", lwd = 3)
```

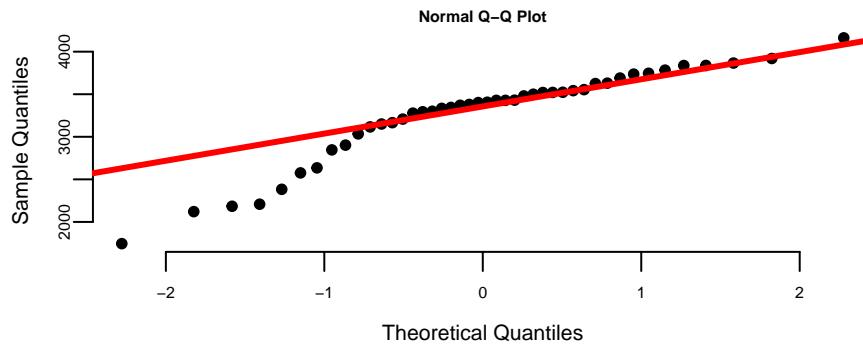


Figure 16.1: Normal Q-Q plot of baby weights

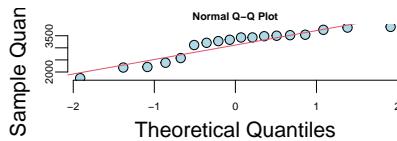
Visual inspection of the Q-Q plot strongly suggests that birth weights are not normally distributed. We can see that on the left side of the plot the points are below the line. This indicates that the data is skewed to the left. The data is not normally distributed.

The Q-Q plots look different if we split the data based on the gender

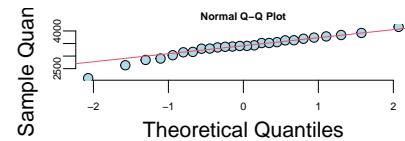
```

par(mar = c(4, 4, 0.7, 0.5), bty = "n", cex.lab = 1, cex.axis =
  0.5, cex.main = 0.5, pch = 21, cex = 1.3)
g <- babyboom %>%
  filter(gender == "girl") %>%
  pull(wt)
b <- babyboom %>%
  filter(gender == "boy") %>%
  pull(wt)
qqnorm(g, bg = "lightblue")
qqline(g, col = 2)
qqnorm(b, bg = "lightblue")
qqline(b, col = 2)

```



(a) Girls
Histogram of baby weights by gender



(a) Boys

How about the times in hours between births of babies?

```

hr <- ceiling(babyboom$running.time / 60)
BirthsByHour <- tabulate(hr)
# Number of hours with 0, 1, 2, 3, 4 births
ObservedCounts <- table(BirthsByHour)
# Average number of births per hour
BirthRate <- sum(BirthsByHour) / 24
# Expected counts for Poisson distribution
ExpectedCounts <- dpois(0:4, BirthRate) * 24
# bind into matrix for plotting
ObsExp <- rbind(ObservedCounts, ExpectedCounts)
barplot(ObsExp, names = 0:4, beside = TRUE, legend =
  c("Observed", "Expected"))

```

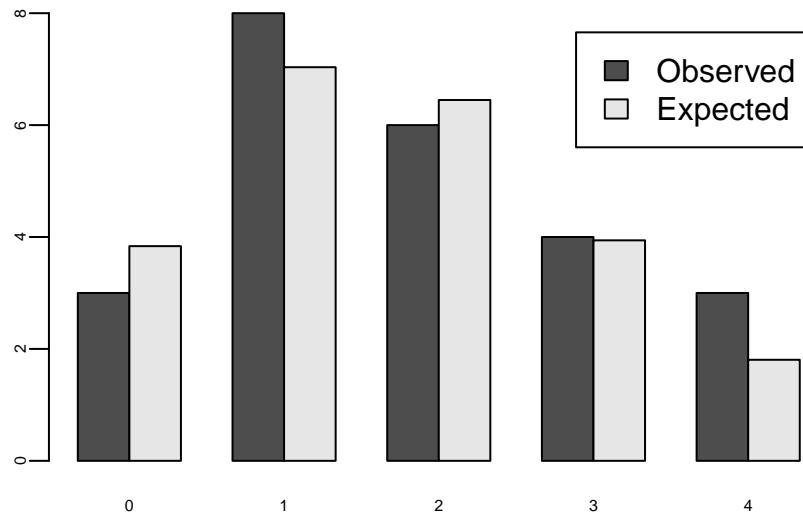
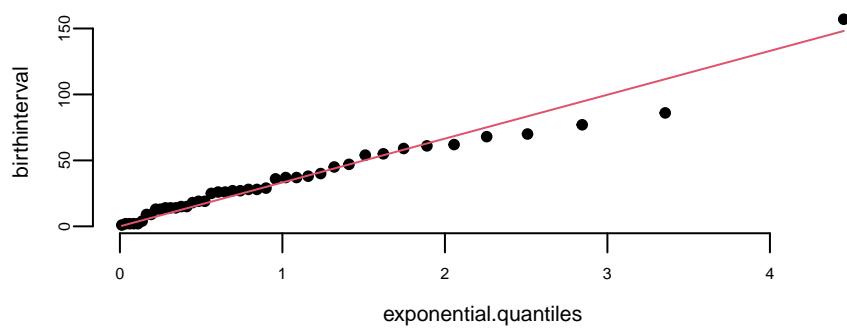


Figure 16.4: Histogram of births by hour

What about the Q-Q plot?

```
# birth intervals
birthinterval <- diff(babyboom$running.time)
# quantiles of standard exponential distribution (rate=1)
exponential.quantiles <- qexp(ppoints(43))
qqplot(exponential.quantiles, birthinterval, bg = "lightblue")
lmb <- mean(birthinterval)
lines(exponential.quantiles, exponential.quantiles * lmb, col =
  2) # Overlay a line
```



Here

- `ppoints` function computes the sequence of probability points
 - `qexp` function computes the quantiles of the exponential distribution
 - `diff` function computes the difference between consecutive elements of a vector
-

16.4 Out-of-Sample Performance

A parametric model is selected from a family of functions, with optimization finding the best member by minimizing empirical loss or maximizing likelihood. Finding the right family is the *model selection* problem: which predictors, interactions, or transformations achieve the best balance between complexity and accuracy? In practice, several models often perform nearly identically.

The optimal model is rarely the one that fits training data perfectly—this typically indicates overfitting, where the model captures noise rather than signal. A good model balances fit against simplicity to ensure generalization. Including too many parameters can yield perfect in-sample fit, but poor out-of-sample performance.

The goal of model selection is not only to achieve a good fit but also to reduce complexity by excluding unnecessary parameters. This process typically involves selecting a model from a relevant class of functions while keeping in mind the trade-offs between bias, variance, and model complexity. Techniques such as cross-validation, information criteria (e.g., AIC, BIC), and regularization methods are commonly used to guide the model selection process.

The model selection task is sometimes one of the most time-consuming parts of data analysis. Unfortunately, there is no single rule to find the best model. One way to think about the model choice problem is as yet another optimization problem, with the goal of finding the best family of functions that describe the data. With a small number of predictors, we can use brute force (check all possible models). For example, with p predictors there are 2^p possible models with no interactions. Thus, the number of potential function families is huge even for modest values of p . One cannot consider all transformations and interactions.

Our goal is to build a model that predicts well for out-of-sample data, i.e., data that was not used for training. Ultimately, we are interested in using our models for prediction, and thus out-of-sample performance is the most important metric and should be used to choose the final model. In-sample performance is of little interest when choosing a predictive model, as one of

the winners of the Netflix prize put it: “It’s like predicting how much someone will like a movie, having them watch it and tell you how much they really liked it.” Out-of-sample performance is the final judge of the quality of our model. The goal is to use data to find a pattern that we can exploit. The pattern will be “statistical” in nature. To uncover the pattern, we start with a training dataset, denoted by

$$D = (y_i, x_i)_{i=1}^n$$

and to test the validity of our model, we use an out-of-sample testing dataset

$$D^* = (y_j^*, x_j^*)_{j=1}^m,$$

where x_i is a set of p predictors and y_i is the response variable.

A good predictor will “generalize” well and provide low MSE out-of-sample. There are a number of methods/objective functions that we will use to find \hat{f} . In a parameter-based approach, we will find a black box. There are a number of ways to build our black box model. Our goal is to find the map f that approximates the process that generated the data. For example, data could represent some physical observations, and our goal is to recover the “laws of nature” that led to those observations. One of the pitfalls is to find a map f that does not generalize. Generalization means that our model actually learned the “laws of nature” and not just identified patterns present in training. The lack of generalization of the model is called overfitting. It can be demonstrated in one dimension by remembering the fact from calculus that any set of n points can be approximated by a polynomial of degree n , e.g., we can always draw a line that connects two points. Thus, in one dimension we can always find a function with zero empirical risk. However, such a function is unlikely to generalize to observations that were not in our training data. In other words, the empirical risk measure for D^* is likely to be very high. Let us illustrate that in-sample fit can be deceiving.

Example 16.2 (Hard Function). Say we want to approximate the following function

$$f(x) = \frac{1}{1 + 25x^2}.$$

This function is simply a ratio of two polynomial functions and we will try to build a linear model to reconstruct this function

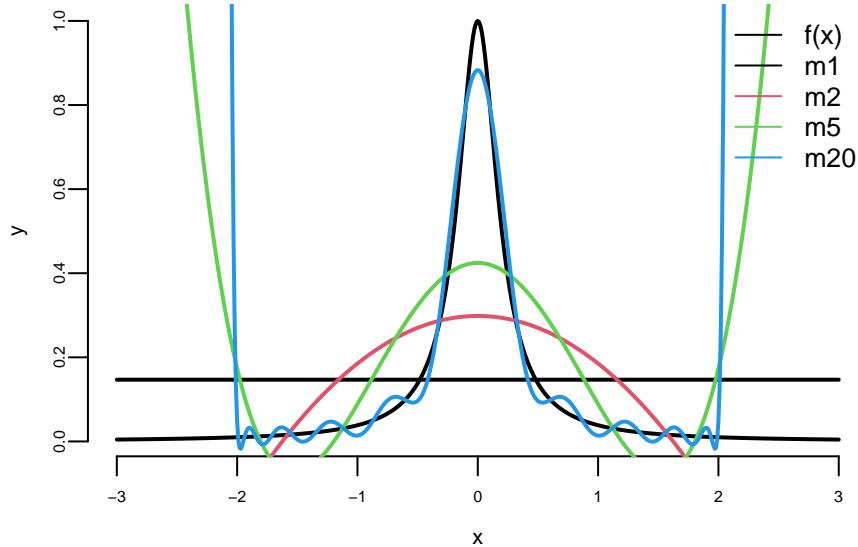


Figure 16.5: Runge-Kutta function

Figure 16.5 shows the function itself (black line) on the interval $[-3, 3]$. We used observations of x from the interval $[-2, 2]$ to train the data (solid line) and from $[-3, -2] \cup (2, 3]$ (dotted line) to test the model and measure the out-of-sample performance. We tried four different linear functions to capture the relations. We see that linear model $\hat{y} = \beta_0 + \beta_1 x$ is not a good model. However, as we increase the degree of the polynomial to 20, the resulting model $\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_{20} x^{20}$ does fit the training dataset quite well, but does a very poor job on the test dataset. Thus, while in-sample performance is good, the out-of-sample performance is unsatisfactory. We should not use the degree 20 polynomial function as a predictive model. In practice, in-sample loss or classification rates provide us with a metric for comparing different predictors. It is worth mentioning here that there should be a penalty for overly complex rules that fit extremely well in-sample but perform poorly on out-of-sample data. As Einstein famously said, “A model should be simple, but not simpler.”

To a Bayesian, the solutions to these decision problems are rather obvious: compute posterior distributions, and then make decisions by maximizing expected utility, where the posterior distribution is used to calculate the expectations. Classical solutions to these problems are different, and use repeated sampling ideas, whereby the performance of a decision rule is judged on its performance if the same decision problem were repeated infinitely. Thus, the decisions are made based on their population properties. One of the main uses of statistical decision theory is to compare different estimators or hypothesis

testing procedures. This theory generates many important findings, most notably that many of the common classical estimators are “bad”, in some sense, and that Bayesian estimators are always “good”.

These results have major implications for empirical work and practical applications, as they provide a guide for forecasting.

16.5 Bias-Variance Trade-off

For any predictive model, we seek to achieve the best possible results, i.e., the smallest MSE or misclassification rate. For a historical perspective on how aggregating independent estimates reduces variance while preserving low bias, see the discussion of Galton’s ox-weighing experiment in Section 10.1.3. However, model performance can vary depending on the specific training/validation split used. A model that performed well on one test set may not produce good results given additional data. Sometimes we observe situations where a small change in the data leads to a large change in the final estimated model, e.g., the parameters of the model. These results exemplify the bias/variance tradeoff, where increasing model bias can reduce variance in the final results. Similarly, low bias can result in high variance, but can also produce an oversimplification of the final model. The bias/variance concept is depicted below.

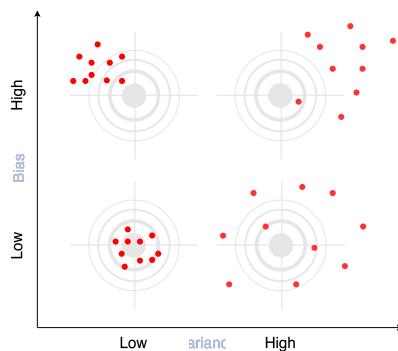


Figure 16.6: Bias-variance trade-off

Example 16.3 (Bias-variance). We demonstrate bias-variance concept using Boston housing example. We fit a model $\text{medv} = f(\text{lstat})$. We use polynomial functions to approximate this relation. We fitted twelve polynomial functions with degree 1, ..., 12 ten time. Each time we randomly selected 20% of sample for testing and the rest for training. We estimated in-of-sample performance

(bias) and out-of-sample performance by calculating MSE on training and testing sets correspondingly. For each polynomial f we averaged MSE from each of the ten models.

Figure 16.7a shows bias and variance for our twelve different models. As expected, bias increases while variance decreases as model complexity grows. On the other hand out-of-sample MSE is a U-shaped curve. The optimal model is the one that has smallest out-of-sample MSE. In our case it is polynomial of degree 5!

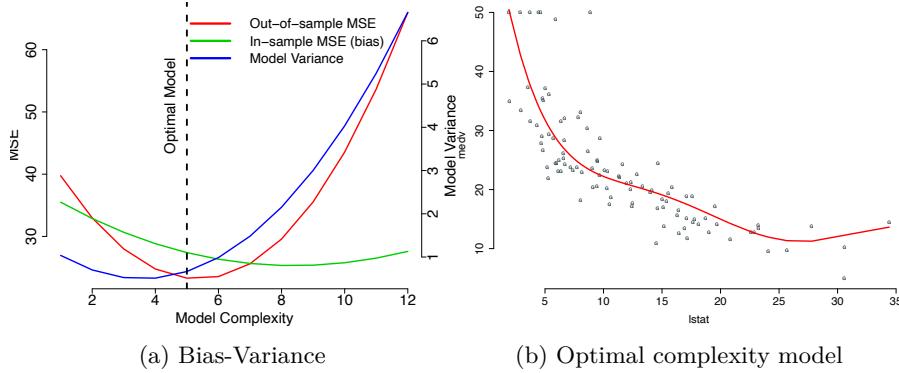


Figure 16.7: Metrics for twelve polynomial functions fitted into Boston housing data set. Left panel: As model complexity (degree of the polynomial function) increases, model variance increase and bias decreases. Out-of-sample MSE is smallest for 5th degree polynomial function, which is the optimal model in terms of bias-variance trade-off. Right panel: Optimal model, which is 5th degree polynomial used to predict observations from testing data set. Model predictions (red line) are compared to actual observed values of medv variable (dots).

Let's take another, more formal look at the bias-variance trade-off for a linear regression problem. We are interested in the decomposition of the error $E((y - \hat{y})^2)$ as a function of bias $E(y - \hat{y})$ and variance $\text{Var}(\hat{y})$.

Here $\hat{y} = \hat{f}_\beta(x)$ is the prediction from the model, and $y = f(x) + \epsilon$ is the true value, which is measured with noise $\text{Var}(\epsilon) = \sigma^2$, where $f(x)$ is the true unknown function. The expectation above measures the squared error of our

model on a random sample x .

$$\begin{aligned}
 E((y - \hat{y})^2) &= E(y^2 + \hat{y}^2 - 2y\hat{y}) \\
 &= E(y^2) + E(\hat{y}^2) - E(2y\hat{y}) \\
 &= \text{Var}(y) + E(y)^2 + \text{Var}(\hat{y}) + E(\hat{y})^2 - 2fE(\hat{y}) \\
 &= \text{Var}(y) + \text{Var}(\hat{y}) + (f^2 - 2fE(\hat{y}) + E(\hat{y})^2) \\
 &= \text{Var}(y) + \text{Var}(\hat{y}) + (f - E(\hat{y}))^2 \\
 &= \sigma^2 + \text{Var}(\hat{y}) + \text{Bias}(\hat{y})^2
 \end{aligned}$$

Here we used the following identity: $\text{Var}(X) = E(X^2) - E(X)^2$ and the fact that f is deterministic and $E(\epsilon) = 0$, thus $E(y) = E(f(x) + \epsilon) = f + E(\epsilon) = f$.

16.6 Cross-Validation

If the dataset at hand is small and we cannot dedicate a large enough sample size for testing, simply measuring error on a test dataset can lead to wrong conclusions. When the size of the testing set D^* is small, the estimated out-of-sample performance has high variance, depending on precisely which observations are included in the test set. On the other hand, when the training set D^* is a large fraction of the entire sample available, the estimated out-of-sample performance will be underestimated. Why?

A simple solution is to perform the training/testing split randomly several times and then use the average out-of-sample errors. This procedure has two parameters: the fraction of samples to be selected for testing p and the number of estimates to be performed K . The resulting algorithm is as follows:

```

fsz = as.integer(p*n)
error = rep(0,K)
for (k in 1:K)
{
  test_ind = sample(1:n, size = fsz)
  training = d[-test_ind,]
  testing = d[test_ind,]
  m = lm(y~x, data=training)
  yhat = predict(m, newdata = testing)
  error[k] = mean((yhat-testing$y)^2)
}
res = mean(error)

```

Figure 16.8 shows the process of splitting the dataset randomly five times.

Cross-validation modifies the random splitting approach to use a more “disciplined” way to split the dataset for training and testing. Instead of randomly selecting training data points, CV chooses consecutive observations, and thus each data point is used once for testing. Like the random approach, CV helps address the high variance issue of out-of-sample performance estimation when the available dataset is small. Figure 16.9 shows the process of splitting the dataset five times using the cross-validation approach.

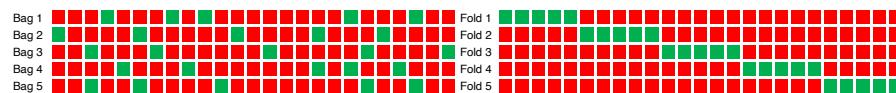


Figure 16.8: Bootstrap
Training set (red) and testing set
(green)

Figure 16.9: Cross-validation

Example 16.4 (Simulated). We use simulated data set to demonstrate difference between estimated out-of-sample performance using random 20/80 split, 5-fold cross-validation and random split. We used $x = -2, -1.99, -1.98, \dots, 2$ and $y = 2 + 3x + \epsilon$, $\epsilon \sim N(0, \sqrt{3})$. We simulated 35 datasets of size 100. For each of the simulated data sets, we fitted a linear model and estimated out-of-sample performance using three different approaches. Figure 16.10 compares empirical distribution of errors estimated from 35 samples.

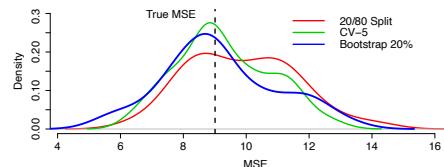


Figure 16.10: Empirical comparison of simple split, cross-validation, and bootstrap approaches to estimate out-of sample performance.

As we can see the estimated out-of-sample performance by a training set approach is of high variance. While, both cross-validation and bootstrap approaches lead to better estimates, they require model to be fitted 5 times, which can be computationally costly for a complex model. On the other hand, estimate from cross-validation is of lower variance and less bias compared to the bootstrap estimate. Thus, we should prefer cross-validation.

16.7 Model evaluation: calibration, discrimination, and scoring

Variable selection is often presented as choosing a subset of predictors. A complementary view is that we are searching for a low-dimensional summary of the inputs that retains predictive information: an informal analogue of the idea of a minimal sufficient statistic from classical inference (Chapter 3).

When we evaluate models, it is useful to separate three questions. First, does the model rank or separate cases well (discrimination)? Second, do the model's predicted probabilities match observed frequencies (calibration)? Third, when we must make decisions, how should we score probabilistic forecasts so that "better" means "closer to the truth" in a principled way?

For binary outcomes, discrimination is often summarized by the receiver operating characteristic (ROC) curve. For a thresholded score, define the true positive rate (TPR) and false positive rate (FPR) as the threshold varies. The ROC curve plots TPR against FPR across all thresholds, and the area under the curve (AUC) summarizes performance as a single number between 0 and 1, with 0.5 corresponding to random ranking; see Fawcett (2006).

16.7.1 Calibration and calibration plots

Calibration concerns the probabilistic interpretation of predicted probabilities. A well-calibrated model that outputs $\hat{p} = 0.8$ on many cases should be correct about 80% of the time on those cases. A common diagnostic is a calibration plot (reliability diagram), which bins predictions and compares average predicted probability to the empirical frequency in each bin.

16.7.2 Proper scoring rules (and the decision-theory link)

A proper scoring rule assigns a numerical score to a probabilistic forecast and is designed so that, in expectation, the best strategy is to report the true predictive distribution; see Gneiting and Raftery (2007). Two widely used examples are the log score (negative log predictive density) and the Brier score for binary outcomes Brier (1950). These connect directly to Chapter 4: choosing a scoring rule is equivalent to choosing a loss for probabilistic prediction, and the expected score becomes a decision-theoretic risk.

For Bayesian models, posterior predictive checks compare observed data to replicated data drawn from the posterior predictive distribution. The workflow is: sample parameters from the posterior, simulate replicated datasets, and

compare summary statistics or discrepancies between observed and replicated data; see Gelman et al. (2013).

16.8 Bayesian Model Selection

The probabilistic models of interest are the joint probability distribution $p(D, \theta)$ (called a generative model) and $P(Y, \theta | X)$ (discriminative model). Discriminative models are easier to build and are more frequently used in practice. Generative models require modeling a distribution over the set of observed variables, which makes our model more complicated. Text analysis provides an illustrative example. The task of identifying the topic of an article can be solved using a discriminative distribution. The problem of generating a new article requires a generative model.

Let D denote data. Let $\theta_M \in \Theta_M$ denote a set of parameters under model $M \in \mathcal{M}$. Let $\theta_M = (\theta_1, \dots, \theta_M)$ be the p -vector of parameters. The Bayesian approach is straightforward: implement the Bayesian paradigm by executing Bayes' rule. This requires the laws of probability and not optimization techniques. The notion of model complexity is no different. Let \mathcal{M} denote the space of models and θ be the parameter vector. The Bayesian paradigm simply places probabilities over parameters and models given the data, namely $p(\theta_M, M | y)$, where $y = (y_1, \dots, y_n)$.

This has a number of decompositions. Bayes' theorem calculates the joint posterior over parameters and models given data D , namely

$$p(\theta_M, M | D) = p(\theta_M | M, D)P(M | D).$$

Notice how this factors the posterior into two terms: the conditional posterior over parameters given the model and the posterior over models given data.

The key quantity is the weight of *evidence* (a.k.a. marginal distribution of the data D given the model M), defined by

$$p(D | M) = \int_{\Theta_M} p(D | \theta_M, M)p(\theta_M | M)d\theta_M.$$

Here $p(D | \theta_M, M)$ is the traditional likelihood function. The key conditional distribution, however, is the specification of the prior over parameters $p(\theta_M | M)$. As this is used in the marginalization, it can affect the Bayes risk dramatically. Occam's razor comes from the fact that this marginalization provides a weight of evidence that favors simpler models over more complex ones.

This leads to a posterior over models, which is calculated as:

$$P(M | D) = \frac{p(D | M)P(M)}{p(D)},$$

$$p(D | M) = \int_{\Theta_M} p(D | \theta_M, M)p(\theta_M | M)d\theta_M.$$

Notice that this requires a joint prior specification $p(\theta_M, M) = p(\theta_M | M)p(M)$ over parameters and models. The quantity $p(M|D)$ is the marginal posterior for model complexity given the data. There is an equivalent posterior $p(\theta_M | D)$ for the parameters. $p(D | M)$ is the evidence of the data D given the complexity (a.k.a. conditional likelihood). The full evidence is

$$p(D) = \int p(D|M)p(M)dM.$$

This has been used to select the amount of hyperparameter regularization; see, for example, MacKay (1992).

We will see that the prior $p(\theta_M | M)$ will lead to an Occam's razor effect, namely that the marginal distribution will favor simpler models. Importantly, this Occam's razor effect is not in conflict with the Bayesian double descent phenomenon, which emerges from the marginal posterior of models given data and the conditional prior specification $p(\theta_M | M)$.

Example 16.5 (Dice Example). Let's consider a simple example of throwing a dice. Say, there are three dice, one that has numbers 1 through 6 (regular dice), one with three sides with ones and three sides with 2s (dice 1-2), one with three sides with ones and three sides with 2s and three sides with 3s (dice 1-2-3).



(a) Regular Dice



(a) Dice 1-2



(a) Dice 1-2-3

You observe outcome of one dice throw and it is 3. Which dice is it?

Two out of three explanations are plausible (dice 1-2-3 and regular dice). Intuitively, the 1-2-3 dice is more likely to produce 3 than the regular dice. Thus, if we need to choose, we would choose the 1-2-3 dice. For the sake of completeness, we can use the Bayes rule to calculate the evidence for each model.

Using Bayes' rule:

$$P(M_i | D) = \frac{P(D | M_i)P(M_i)}{P(D)}$$

where M_i represents each dice model and D is the observed data (outcome = 3).

We set equal prior probabilities for each dice:

$$P(M_1) = P(M_2) = P(M_3) = \frac{1}{3}.$$

Now, we calculate the likelihood for each model.

Dice Type	Probability of Rolling a 3
Regular dice (M_1)	$P(3 M_1) = 1/6$
Dice 1-2 (M_2)	$P(3 M_2) = 0$
Dice 1-2-3 (M_3)	$P(3 M_3) = \frac{1}{3}$

Then, the marginal likelihood is:

$$P(D) = \sum_{i=1}^3 P(D|M_i)P(M_i) = \frac{1}{6} \cdot \frac{1}{3} + 0 \cdot \frac{1}{3} + \frac{1}{3} \cdot \frac{1}{3} = \frac{1}{6}.$$

Finally, posterior probabilities are

Dice Type	Posterior Probability
Regular dice	$P(M_1 D) = \frac{\frac{1}{6} \cdot \frac{1}{3}}{\frac{1}{6}} = 1/3$
Dice 1-2	$P(M_2 D) = \frac{0 \cdot \frac{1}{3}}{\frac{1}{6}} = 0$
Dice 1-2-3	$P(M_3 D) = \frac{\frac{1}{3} \cdot \frac{1}{3}}{\frac{1}{6}} = 2/3$

Given the observation of outcome 3, the dice 1-2-3 is twice as likely as the regular dice. The dice 1-2 is completely ruled out since it cannot produce a 3. This demonstrates how Bayesian model selection naturally eliminates impossible explanations and provides relative evidence for competing hypotheses.

This example demonstrates how the Bayesian paradigm provides a coherent framework to simultaneously infer parameters and model complexity. The fact that Bayesian approach selects the most probable model that explains the observed data, is called the automatic Occam's razor. The Occam's razor is a principle that states that the simplest explanation is the best explanation.

While performing data analysis using learning algorithms, we perform two tasks, namely training and inference which are summarized in the table below

Step	Given	Hidden	What to find
Training	$D = (X, Y) = \{x_i, y_i\}_{i=1}^n$	θ	$p(\theta D)$
Prediction	x_{new}	y_{new}	$p(y_{\text{new}} x_{\text{new}}, D)$

The training can be performed via the Bayes rule

$$p(\theta | D) = \frac{p(Y | \theta, X)p(\theta)}{\int p(Y | \theta, X)p(\theta)d\theta}.$$

Now to perform the second step (prediction), we calculate

$$p(y_{\text{new}} | x_{\text{new}}, D) = \int p(y_{\text{new}} | x_{\text{new}}, \theta)p(\theta | D)d\theta$$

Thus, full Bayesian inference requires calculating two integrals, which might be difficult. We mentioned earlier that MAP allows us to avoid those calculations by approximating the posterior with

$$p(\theta | D) \approx \delta(\theta_{\text{MAP}}), \quad \theta_{\text{MAP}} \in \arg \max_{\theta} p(\theta | D)$$

To calculate θ_{MAP} , we do not need to know the normalizing constant for calculating posterior, since the solution of optimization problem does not depend on this constant. Further, the second integral for inference becomes degenerate and get approximated by

$$p(y_{\text{new}} | x_{\text{new}}, D) = \int p(y_{\text{new}} | x_{\text{new}}, \theta)p(\theta | D)d\theta \approx p(y_{\text{new}} | x_{\text{new}}, \theta_{\text{MAP}}).$$

The Figure 16.14 below illustrates the Bayesian model selection process. The figure shows the joint distribution over parameters and data for three models. You can think of each ellipse as the region where most of the probability mass is concentrated.

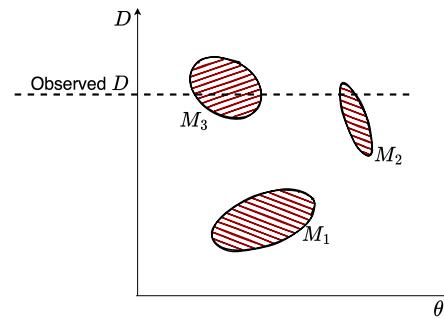


Figure 16.14: Bayesian model Selection

If we project the ellipses onto the parameter space, we get the prior distributions for each model. We can see that the M_2 is the most concentrated. If we project the ellipses onto the data space, we get the prior distributions over data for each model.

After observing data D (horizontal line), each prior gets updated. The intersection of the observed data line with each ellipse shows how well each model can explain the data. Models with good overlap between prior and observed data will have higher posterior probability. M_3 appears to have the best intersection with the observed data, it is the model with the highest marginal likelihood.

This illustrates how Bayesian model selection naturally favors models that achieve the best balance between explaining the observed data and maintaining appropriate complexity, automatically implementing Occam's razor through the evidence calculation.

Example 16.6 (Racial discrimination). Say we want to analyze racial discrimination by the US courts. We have three variables:

- Murderer: $m \in \{0, 1\}$ (black/white)
- Victim: $v \in \{0, 1\}$ (black/white)
- Verdict: $d \in \{0, 1\}$ (prison/death penalty)

Say we have the data

m	v	d	n
0	0	0	132
0	0	1	19
0	1	0	9
0	1	1	0
1	0	0	52
1	0	1	11
1	1	0	97
1	1	1	6

We would like to establish a causal relations between the race and verdict variables. For this, we consider several models

1. $p(d | m, v) = p(d) = \theta$
2. $p(d | m, v) = p(d | v); p(d | v = 0) = \alpha, p(d | v = 1) = \beta$
3. $p(d | v, m) = p(d | m); p(d | m = 0) = \gamma, p(d | m = 1) = \delta$

4. $p(d|v, m)$ cannot be reduced, and

$p(d = 1 m, v)$	$m = 0$	$m = 1$
$v = 0$	τ	χ
$v = 1$	ν	ζ

We calculate which model describes data the best, we calculate the evidences. We need to describe the discriminative model

$$p(Y, \theta | X) = p(Y | X, \theta)p(\theta | X)$$

Here X is the number of cases, and Y is the number of death penalties. We use uninformative prior $\theta \sim U[0, 1]$. To specify the likelihood, we use Binomial distribution

$$Y | X, \theta \sim \text{Bin}(X, \theta), \quad \text{Bin}(Y | X, \theta) = \binom{X}{Y} \theta^Y (1 - \theta)^{X-Y}$$

We assume $p(\theta) \sim \text{Uniform}$. Now lets calculate the evidence

$$p(Y, \theta | X) = \int p(Y | X, \theta)p(\theta)d\theta$$

for each of the four models

1. $p(Y | X) = \int \text{Bin}(19 | 151, \theta)\text{Bin}(0 | 9, \theta)\text{Bin}(11 | 63, \theta)\text{Bin}(6 | 103, \theta)d\theta \propto \int_0^1 \theta^{36}(1 - \theta)^{290}d\theta = \text{Beta}(37, 291) = 2.8 \times 10^{-51}$
2. $p(Y | X) = \int \int \text{Bin}(19 | 151, \alpha)\text{Bin}(0 | 9, \beta)\text{Bin}(11 | 63, \alpha)\text{Bin}(6 | 103, \beta)d\alpha d\beta \propto 4.7 \times 10^{-51}$
3. $p(d | v, m) = p(d | m) = \int \int \text{Bin}(19 | 151, \gamma)\text{Bin}(0 | 9, \gamma)\text{Bin}(11 | 63, \delta)\text{Bin}(6 | 103, \delta)d\gamma d\delta \propto 0.27 \times 10^{-51}$
4. $p(d | v, m) = \int \int \int \text{Bin}(19 | 151, \tau)\text{Bin}(0 | 9, \nu)\text{Bin}(11 | 63, \chi)\text{Bin}(6 | 103, \zeta)d\tau d\nu d\chi \propto 0.18 \times 10^{-51}$

The last model (4) is overly complex. With enough parameters, it can perfectly fit any dataset, including the noise. Ideally, we want a model that fits the data well but is also parsimonious. The Bayesian evidence calculation naturally penalized the complex model for its large parameter space, resulting in the lowest evidence score. This illustrates how the Bayesian approach inherently protects against overfitting without requiring an explicit regularization term.

This dataset also illustrates **Simpson's paradox**, a phenomenon where a trend appears in different groups of data but disappears or reverses when these groups are combined. This highlights the importance of considering confounding variables and the structure of the data. A related conceptual problem is **Bertrand's box paradox**, which demonstrates how conditional probability can be counterintuitive.

16.8.1 Bayesian Information Criterion (BIC)

The Bayesian Information Criterion (BIC) is a model selection criterion that penalizes the complexity of the model. It is derived from a Bayesian approach. The BIC is defined as:

$$\text{BIC} = \log p(D | \hat{\theta}_k, M_k) - \frac{k}{2} \log n.$$

Here $\hat{\theta}_k$ is the MAP estimate of the k parameters in model M_k , and n is the sample size. As such, there is a penalty $-\frac{k}{2} \log n$ for increasing the dimensionality k of the model under consideration.

The BIC uses the marginal likelihood of the data under model M_k (denoted M for simplicity here), which is approximated using Laplace's method.

The idea of Laplace's method is to approximate integrals of the form $\int f(\theta) e^{-g(\theta)} d\theta$ where $g(\theta)$ has a sharp minimum at some point $\hat{\theta}$. The method works by approximating $g(\theta)$ with its second-order Taylor expansion around the minimum $\hat{\theta}$. Since $g'(\hat{\theta}) = 0$ at the minimum, we have

$$g(\theta) \approx g(\hat{\theta}) + \frac{1}{2}g''(\hat{\theta})(\theta - \hat{\theta})^2.$$

So the integral transforms into a Gaussian form:

$$\int f(\theta) e^{-g(\theta)} d\theta \approx f(\hat{\theta}) e^{-g(\hat{\theta})} \int e^{-\frac{1}{2}g''(\hat{\theta})(\theta - \hat{\theta})^2} d\theta.$$

The remaining integral is a standard Gaussian integral that evaluates to $\sqrt{\frac{2\pi}{g''(\hat{\theta})}}$, giving us:

$$\int f(\theta) e^{-g(\theta)} d\theta \approx f(\hat{\theta}) e^{-g(\hat{\theta})} \sqrt{\frac{2\pi}{g''(\hat{\theta})}}.$$

In the multivariate case, we have $\theta \in \mathbb{R}^k$ is a k -dimensional parameter vector. The second-order Taylor expansion around the minimum $\hat{\theta}$ becomes:

$$g(\theta) \approx g(\hat{\theta}) + \frac{1}{2}(\theta - \hat{\theta})^T \mathbf{H}(\hat{\theta})(\theta - \hat{\theta}),$$

where $\mathbf{H}(\hat{\theta})$ is the $k \times k$ Hessian matrix of second derivatives at $\hat{\theta}$. The multivariate Gaussian integral then evaluates to:

$$\int f(\theta) e^{-g(\theta)} d\theta \approx f(\hat{\theta}) e^{-g(\hat{\theta})} (2\pi)^{k/2} |\det(\mathbf{H}(\hat{\theta}))|^{-\frac{1}{2}}$$

In the context of Bayesian model selection, we apply this to approximate the marginal likelihood (evidence). We have:

$$P(D | M) = \int P(D | \theta, M)P(\theta | M)d\theta$$

Taking the logarithm and identifying $g(\theta) = -\log P(D | \theta, M)P(\theta | M)$, the maximum a posteriori (MAP) estimate $\hat{\theta}$ corresponds to the minimum of $g(\theta)$. The second derivative (Hessian) $\mathbf{H}(\hat{\theta})$ at this point determines the curvature of the log-posterior.

$$p(D | M) = \int p(D | \theta, M)p(\theta | M)d\theta \approx p(D | \hat{\theta}, M)p(\hat{\theta} | M)(2\pi)^{k/2}|\det(\mathbf{H}(\hat{\theta}))|^{-\frac{1}{2}}.$$

Here $\hat{\theta}$ is the posterior mode (MAP estimate), and $\mathbf{H}(\hat{\theta})$ is the negative Hessian of the log-posterior at the mode. Taking the logarithm, and assuming $P(\hat{\theta}|M)$ and Hessian terms are $O_p(1)$ or scale appropriately with n (this assumption is justified because as n increases, the likelihood dominates the prior, making the prior term negligible relative to the $O(\log n)$ likelihood term, while the Hessian determinant typically grows polynomially in n , contributing at most $O(\log n)$ terms that are absorbed into the approximation), we get:

$$\log p(D | M) \approx \log p(D | \hat{\theta}, M) - \frac{k}{2} \log n,$$

which is proportional to the BIC. (Note: The exact definition and derivation of BIC can vary slightly, but this captures the essence). The BIC approximation shows how the Bayesian approach naturally penalizes model complexity through the dimensionality term $-\frac{k}{2} \log n$.

The Bayesian approach averages over the posterior distribution of models given data. Suppose that we have a finite list of models $M \in \{M_1, \dots, M_J\}$. Then we can calculate the posterior over models as:

$$p(M_j | y) = \frac{p(y|M_j)p(M_j)}{\sum_{i=1}^J p(y|M_i)p(M_i)}, \quad \text{where } p(y|M_j) = \int L_j(\theta_j | y)p(\theta_j | M_j)d\theta_j.$$

Laplace's approximation provides a simple (Lindley 1961) illustration of how dimensionality is weighted in the Bayesian paradigm. Hence, BIC is related to a log-posterior approximation. Hence, if prior model probabilities $P(M_j)$ are uniform, then $P(M_j | D) \propto P(D | M_j) \approx \exp(\text{BIC}_j)$.

In a more general case, the evidence (a.k.a. marginal likelihood) for hypotheses (a.k.a. models) M_i is calculated as follows:

$$P(D | M_i) = \int P(D | \theta, M_i)P(\theta | M_i)d\theta.$$

Laplace approximation, in the one-dimensional case ($k = 1$), yields:

$$P(D | M_i) \approx P(D | \hat{\theta}, M_i)P(\hat{\theta} | M_i)\sqrt{2\pi}\sigma_{\text{post}}.$$

Here $\hat{\theta}$ is the maximum *a posteriori* (MAP) estimate of the parameter and $\sigma_{\text{post}} = (-H(\hat{\theta}))^{-1/2}$ where $H(\hat{\theta})$ is the second derivative of the log-posterior at $\hat{\theta}$.

Generally, in the k -dimensional case, we have:

$$P(D | M_i) \approx P(D | \hat{\theta}, M_i)P(\hat{\theta} | M_i)(2\pi)^{k/2}|\det(-\mathbf{H}(\hat{\theta}))|^{-\frac{1}{2}}.$$

Here $\mathbf{H}(\hat{\theta}) = \nabla^2 \log(P(D | \hat{\theta}, M_i)P(\hat{\theta} | M_i))$ is the Hessian of the log-posterior function evaluated at the mode $\hat{\theta}$. As the amount of data collected increases, this Gaussian approximation is expected to become increasingly accurate.

Mackay (MacKay 1992) proposes the NIC criterion for selection of neural networks.

16.8.2 Neural Information Criterion (NIC)

David MacKay's Neural Information Criterion (NIC) applies Bayesian model selection to neural networks by computing approximations to the marginal likelihood for different architectures and hyperparameter configurations.

The evidence framework builds upon the Laplace approximation to estimate the marginal likelihood of neural network models. Given a neural network with parameters θ and hyperparameters α (such as weight decay parameters), the evidence for a particular model configuration is:

$$P(D|M, \alpha) = \int P(D|\theta, M)P(\theta|M, \alpha)d\theta$$

where D represents the training data, M denotes the model architecture, and $P(\theta|M, \alpha)$ is the prior distribution over network weights. The Laplace approximation evaluates this integral by expanding the log-posterior around its mode $\hat{\theta}$, yielding:

$$\log P(D|M, \alpha) \approx \log P(D|\hat{\theta}, M) + \log P(\hat{\theta}|M, \alpha) - \frac{1}{2} \log |H|$$

where H is the Hessian of the negative log-posterior at the mode $\hat{\theta}$. This approximation transforms the intractable integral into a computation involving the maximum a posteriori (MAP) estimate and the curvature of the posterior at that point.

The evidence framework provides a principled approach to several critical decisions in neural network design. For hyperparameter selection, the framework automatically determines optimal regularization strengths by maximizing the evidence with respect to hyperparameters such as weight decay coefficients. Rather than relying on cross-validation, which can be computationally expensive and may not capture the full uncertainty in hyperparameter selection, the evidence provides a direct measure of how well different hyperparameter values support the observed data.

Architecture comparison becomes feasible through direct evidence computation for different network structures. The framework can compare networks with different numbers of hidden units, layers, or connectivity patterns by evaluating their respective marginal likelihoods. This comparison naturally incorporates Occam's razor, as more complex architectures are penalized through the integration over their larger parameter spaces, unless the additional complexity is justified by substantially improved fit to the data.

The Hessian computation required for the Laplace approximation presents significant computational challenges for modern deep networks with millions or billions of parameters. The full Hessian matrix would be prohibitively large to compute and store explicitly. MacKay's original framework addressed this through various approximation strategies, including the use of automatic relevance determination (ARD) priors that allow the network to effectively prune irrelevant connections by driving their associated precision parameters to infinity.

16.8.3 Automatic Relevance Determination (ARD)

The key insight of *Automatic Relevance Determination (ARD)* is to introduce separate precision hyperparameters for different groups of parameters, allowing the model to automatically determine which features or components are relevant for the task.

In the context of neural networks, consider a network with weights $\mathbf{w} = \{w_{ij}\}$ connecting input features to hidden units. Instead of using a single precision parameter α for all weights, ARD introduces feature-specific precision parameters $\{\alpha_i\}_{i=1}^p$ where p is the number of input features. The prior distribution for weights becomes:

$$p(\mathbf{w}|\alpha) = \prod_{i=1}^p \prod_{j=1}^H \mathcal{N}(w_{ij}|0, \alpha_i^{-1})$$

where H is the number of hidden units and $\alpha = (\alpha_1, \dots, \alpha_p)$ are the precision hyperparameters.

The hierarchical Bayesian model is completed by placing hyperpriors on the precision parameters:

$$p(\alpha_i) = \text{Gamma}(\alpha_i | a_i, b_i)$$

where a_i and b_i are shape and rate parameters, often set to small values (e.g., $a_i = b_i = 10^{-6}$) to create weakly informative priors.

The ARD mechanism works through the evidence framework by optimizing the marginal likelihood with respect to the hyperparameters. For a given precision α_i , the effective contribution of feature i to the model evidence can be approximated as:

$$\log p(D|\alpha_i) \approx -\frac{1}{2}\alpha_i \|\mathbf{w}_i\|^2 + \frac{H}{2} \log \alpha_i - \frac{1}{2} \log |\mathbf{A}_i|$$

where \mathbf{w}_i represents all weights associated with feature i , and \mathbf{A}_i is the corresponding block of the Hessian matrix.

When a feature is irrelevant, the optimal precision α_i^* tends to infinity, effectively removing the feature from the model. This occurs because the evidence balances the model fit (first term) against the model complexity (second and third terms). For irrelevant features, the improvement in fit is insufficient to justify the complexity cost, driving α_i to large values.

The ARD update equations, derived by maximizing the marginal likelihood, are:

$$\alpha_i^{\text{new}} = \frac{\gamma_i}{\|\mathbf{w}_i\|^2}$$

where γ_i is the effective number of parameters associated with feature i :

$$\gamma_i = H - \alpha_i \text{Tr}(\mathbf{A}_i^{-1})$$

Here, $\text{Tr}(\mathbf{A}_i^{-1})$ represents the trace of the inverse of the Hessian block corresponding to feature i .

Example: Linear Regression with ARD

Consider a linear regression model with ARD priors:

$$y = \sum_{i=1}^p w_i x_i + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \beta^{-1})$$

with priors:

$$w_i \sim \mathcal{N}(0, \alpha_i^{-1}), \quad i = 1, \dots, p$$

The posterior distribution for the weights is:

$$p(\mathbf{w}|D, \alpha, \beta) = \mathcal{N}(\mathbf{w}|\mu, \Sigma)$$

where:

$$\begin{aligned} &= (\beta \mathbf{X}^T \mathbf{X} + \text{diag}(\alpha))^{-1} \\ \mu &= \beta \mathbf{X}^T \mathbf{y} \end{aligned}$$

The ARD updates become:

$$\alpha_i^{\text{new}} = \frac{1 - \alpha_i \Sigma_{ii}}{\mu_i^2}$$

When α_i becomes very large, the corresponding $\mu_i \approx 0$ and $\Sigma_{ii} \approx 0$, effectively removing feature i from the model.

Example: Neural Network Feature Selection

In a neural network with p input features and H hidden units, ARD can automatically determine which input features are relevant. Suppose we have a dataset with features representing different types of measurements, some of which may be irrelevant for the prediction task.

The network architecture is:

$$\begin{aligned} h_j &= \tanh \left(\sum_{i=1}^p w_{ij} x_i + b_j \right), \quad j = 1, \dots, H \\ y &= \sum_{j=1}^H v_j h_j + c \end{aligned}$$

With ARD priors on input weights:

$$w_{ij} \sim \mathcal{N}(0, \alpha_i^{-1}), \quad \text{for all } j$$

After training with the evidence framework, features with large α_i values (typically $\alpha_i > 10^6$) are considered irrelevant and can be pruned. This automatic

feature selection often reveals that only a subset of the original features are necessary for good predictive performance.

The ARD principle extends beyond feature selection to other forms of model selection, including:

- **Unit pruning:** Using separate precisions for different hidden units to determine optimal network architecture
- **Group selection:** Applying ARD to groups of related parameters (e.g., all weights in a particular layer)
- **Sparse coding:** In dictionary learning, ARD can automatically determine the effective dictionary size

The computational implementation of ARD involves iterating between updating the model parameters (weights) and the hyperparameters (precisions) until convergence. Modern implementations often use variational approximations or sampling methods to handle the computational challenges of the full Bayesian treatment, while maintaining the automatic model selection capabilities that make ARD so valuable in practice.

Modern adaptations of the evidence framework have developed sophisticated methods to handle the computational challenges of contemporary deep learning. For example, linearized Laplace approximations—such as those described by Ritter et al. (2018) (Ritter, Botev, and Barber 2018) and Immer et al. (2021) (Immer et al. 2021)—approximate the neural network through its first-order Taylor expansion around the MAP estimate, reducing the complexity of Hessian computations while maintaining reasonable approximation quality. These linearized approaches are particularly effective for networks that are sufficiently wide or when the posterior is approximately Gaussian.

Kronecker-structured approximations represent another significant advancement, exploiting the structure of neural network computations to factorize the Hessian matrix into more manageable components. By recognizing that gradients in neural networks can be expressed as Kronecker products of activations and error signals, these methods achieve substantial computational savings while preserving much of the information contained in the full Hessian matrix. Singh, Farrell-Maupin, and Faghihi (2024) revisit and advance the Laplace approximation for Bayesian deep learning, addressing its scalability and effectiveness for modern neural networks. The paper introduces new algorithmic and theoretical developments that make Laplace-based Bayesian inference practical for large-scale deep learning tasks. The authors propose efficient algorithms for computing the Laplace approximation in deep neural networks, leveraging block-diagonal and Kronecker-factored structures to approximate the Hessian of the loss function, which enables uncertainty quantification in models with millions of parameters.

On the theoretical side, the paper provides a new analysis of the Laplace approximation in high-dimensional and overparameterized regimes typical of

deep learning. Specifically, the authors derive non-asymptotic error bounds for the Laplace approximation, showing how its accuracy depends on the curvature of the loss landscape and the concentration of the posterior. They analyze the impact of model width and data size on the quality of the Gaussian approximation, and clarify under what conditions the Laplace approximation remains reliable as the number of parameters grows. This theoretical work helps explain when and why Laplace-based uncertainty estimates are trustworthy in modern neural networks, and guides the design of scalable algorithms for practical Bayesian deep learning.

The evidence framework also naturally handles the multiple scales of uncertainty present in neural networks. Parameter uncertainty captures the uncertainty in individual weight values given the training data, while hyperparameter uncertainty reflects uncertainty about the appropriate level of regularization or architectural choices. Model uncertainty encompasses uncertainty about the fundamental model class or architecture family. The hierarchical Bayesian treatment allows simultaneous reasoning about all these sources of uncertainty within a unified framework.

Despite its theoretical elegance, the evidence framework faces practical limitations in very large-scale applications. The computational requirements of Hessian approximation, even with modern efficient methods, can be substantial for networks with hundreds of millions of parameters. The Laplace approximation itself may be inadequate when the posterior is highly non-Gaussian, which can occur in networks with many local minima or complex loss landscapes.

The enduring value of MacKay's evidence framework lies in its principled approach to the fundamental trade-offs in machine learning model design. By providing a theoretically grounded method for balancing model complexity against data fit, the framework offers insights that remain relevant even as the scale and sophistication of machine learning models continue to evolve. The automatic hyperparameter selection and architecture comparison capabilities of the evidence framework continue to influence contemporary approaches to neural architecture search and automated machine learning.

Yet another approach is the Widely applicable Bayesian Information Criterion (WBIC) proposed by Watanabe (2013). It is a generalization of the traditional BIC that addresses some of its fundamental limitations, particularly when dealing with singular statistical models and complex machine learning architectures. It addresses the problem when BIC's assumptions that the true parameter lies in the interior of the parameter space and that the information matrix is positive definite are violated. These regularity conditions fail for many important models such as neural networks with hidden units, mixture models where the number of components is unknown, tree-based models with unknown structure, and models with parameter constraints or boundaries. Second, BIC requires knowing the effective number of parameters k , which can be ambiguous for complex models. This becomes problematic when dealing

with shared parameters across different parts of the model, regularization that effectively reduces the parameter dimension, or hierarchical structures where the effective dimensionality depends on the data. The challenge of defining the “true” number of parameters in modern machine learning models makes BIC’s penalty term difficult to specify correctly.

16.9 Model Selection and Bayesian Relativity

Bayes’ rule provides only relative evidence between models. Classical approaches try to find absolute truth, but Bayesian model selection is fundamentally comparative. Consider the basic relationship:

$$\frac{p(H_0 | D)}{p(H_1 | D)} = \frac{p(D | H_0) p(H_0)}{p(D | H_1) p(H_1)}$$

When $D = \{T(y) > t_{obs}\}$, the numerator is the p-value. However, this needs to be assessed relative to the p-value under the alternative, which might be even more unlikely! As Sherlock Holmes noted: “When you have eliminated the impossible, whatever remains, however improbable, must be the truth.” Even though an event may be unlikely under H_0 , it could be the best available description given the alternatives.

Fisher recognized this issue: “In scientific inference, a hypothesis is never proved but merely shown to be more or less probable relative to the available alternatives.” The problem with p-values is that they attempt to be an objective measure of model “fit” without considering alternatives. Unlikely events do occur under a “true” model.

16.9.1 Exhaustive vs Non-Exhaustive Hypotheses

A key point is that you can always calculate the relative evidence between two hypotheses. In cases where hypotheses are exhaustive, $p(H_0) + p(H_1) = 1$, we can directly calculate $p(H_0 | D)$ and we obtain the true probability given the data. In general, we have some prior probability left over for a model that is not in the set of models under consideration. But, you can still use the Bayes rule for relative evidence to obtain just a relative ordering:

$$\frac{p(H_0 | D)}{p(H_1 | D)} = \frac{p(D | H_0) p(H_0)}{p(D | H_1) p(H_1)}$$

This holds for $p(H_0) + p(H_1) < 1$. An important benefit of this rational approach is that if a new hypothesis H_2 comes along, the relative calculation be-

tween H_0 and H_1 doesn't change! This is a benefit of rational decision-making. However, posterior probabilities can change if we re-normalize to account for the new alternative.

16.10 The Asymptotic Carrier

What happens when the “true” model is not in the set of models under consideration? This is a critical question in modern machine learning, where model misspecification is the norm rather than the exception.

The asymptotic behavior of the posterior $p(\theta | y)$ is characterized by the *asymptotic carrier* of the posterior. Let F denote the true data-generating process. The set \mathcal{C} is defined by:

$$\mathcal{C} = \arg \min_{\theta \in \Theta} \int f(y) \log f_\theta(y) dy = \arg \min_{\theta \in \Theta} KL(f, f_\theta)$$

That is, the posterior over parameters θ in the model class \mathcal{M} converges to the density that minimizes the Kullback-Leibler (KL) distance between the data-generating process f and the model class.

The posterior has the limiting property that for any $A \subset \mathcal{C}$:

$$\lim_{n \rightarrow \infty} P_{\mathcal{M}}[A | y_1, \dots, y_n] = 1 \text{ almost surely under } F$$

Since Berk (1966), there has been extensive work on the limiting behavior of the posterior when the true model f lies outside the class $\{f_\theta\}$ indexed by the models under consideration. This theory provides important insights:

1. **Consistency under misspecification:** Even when the true model is not in our class, the posterior will concentrate on the best approximation within that class.
2. **KL optimality:** The limiting posterior focuses on parameters that minimize the KL divergence, which is often a reasonable criterion for model approximation.
3. **Practical implications:** This suggests that Bayesian methods can be robust to model misspecification, concentrating probability mass on the best available approximation.

16.11 Model Explainability

In some applications, model explainability is an important criterion for model selection. For example, in finance or insurance, model explainability is important for the model to be accepted by the regulators. It is highly unlikely a black box model will ever be used for medical or criminal justice applications. While earlier in this chapter we introduced several explainability techniques in the context of fundamental considerations, here we provide a comprehensive treatment of model explainability methods, their applications, and a broader perspective on the role of explanation in scientific and practical domains.

Modern machine learning has developed a rich toolkit of methods to interpret and explain model predictions. These methods vary in their scope (global vs. local explanations), their model specificity (model-agnostic vs. model-specific), and their theoretical foundations.

Model-Agnostic Methods provide explanations that work with any machine learning model by treating it as a black box. LIME (Local Interpretable Model-agnostic Explanations) Ribeiro, Singh, and Guestrin (2016) creates explanations for individual predictions by training simple, interpretable models locally around the prediction of interest. The algorithm perturbs the input and observes how the model's predictions change, then fits a linear model weighted by proximity to the original instance. SHAP (SHapley Additive exPlanations) Lundberg and Lee (2017) provides a unified framework based on game theory, specifically Shapley values from cooperative game theory. SHAP assigns each feature an importance value for a particular prediction, satisfying desirable properties like local accuracy, missingness, and consistency. The method has become widely adopted due to its solid theoretical foundation and availability of efficient implementations for various model types.

Partial Dependence Plots (PDP) Jerome H. Friedman (2001) and Individual Conditional Expectation (ICE) plots Goldstein et al. (2015) offer visual methods to understand how features influence predictions across the entire dataset. PDPs show the marginal effect of a feature on the predicted outcome by averaging over all other features, while ICE plots show the prediction dependence for individual instances. Accumulated Local Effects (ALE) Apley and Zhu (2020) improve upon PDPs by accounting for feature correlations, providing more reliable interpretations in high-dimensional spaces.

Gradient-Based Methods leverage the internal structure of differentiable models, particularly neural networks. Saliency maps Simonyan, Vedaldi, and Zisserman (2013) compute gradients of the output with respect to input features to identify which inputs most influence predictions. Integrated Gradients Sundararajan, Taly, and Yan (2017) improve upon simple gradients by integrating gradients along a path from a baseline to the actual input, satisfying important

axioms like sensitivity and implementation invariance. Grad-CAM (Gradient-weighted Class Activation Mapping) Selvaraju et al. (2017) generates visual explanations for convolutional neural networks by computing gradients of the target class with respect to feature maps in the final convolutional layer, producing class-discriminative localization maps that highlight important regions in images.

Attention Mechanisms Bahdanau, Cho, and Bengio (2014) provide built-in interpretability, particularly in natural language processing and computer vision. By learning to focus on relevant parts of the input, attention weights offer direct insight into which components the model considers important for its predictions. Transformer architectures Vaswani et al. (2017) have made attention mechanisms ubiquitous in modern deep learning, though interpreting multi-head attention in large language models remains an active research area Clark et al. (2019).

Model-Specific Interpretability includes methods designed for particular model architectures. Tree-based models like random forests and gradient boosting machines provide natural feature importance measures based on how much each feature decreases impurity or loss across splits Breiman (2001) Jerome H. Friedman (2001). Linear models offer direct coefficient interpretation, though care must be taken with correlated features. Rule extraction methods Craven and Shavlik (1996) attempt to distill complex models into human-readable rule sets.

Counterfactual Explanations Wachter, Mittelstadt, and Russell (2017) answer the question “what would need to change for the prediction to be different?” by finding the minimal modifications to input features that would alter the model’s decision. This approach is particularly valuable in applications like loan decisions, where explaining why an application was rejected is less actionable than explaining what changes would lead to approval.

We now provide detailed mathematical foundations for the most widely-used explainability methods, accompanied by practical demonstrations in R. These methods represent the core toolkit for model interpretation in modern machine learning practice.

16.11.1 The Imperative for Explainability

The demand for explainability stems from multiple sources, each with distinct requirements and motivations. In regulated industries, explainability is often a legal requirement. The European Union’s GDPR includes provisions for algorithmic transparency, requiring that individuals affected by automated decisions receive meaningful information about the logic involved. The Equal Credit Opportunity Act in the United States mandates that financial institutions provide specific reasons for adverse credit decisions. Healthcare appli-

cations face similar regulatory scrutiny, where the FDA increasingly requires evidence of interpretability for AI-assisted diagnostic tools.

Trust and adoption present another critical dimension. Medical practitioners are unlikely to rely on diagnostic systems they don't understand, regardless of their accuracy. A radiologist examining a potential tumor needs to see which image features led to the model's assessment, allowing them to combine algorithmic insights with their clinical expertise. Financial advisors similarly require explanations to justify investment recommendations to clients and comply with fiduciary duties. In these contexts, explainability is not merely desirable but essential for practical deployment.

Debugging and model improvement benefit significantly from interpretability tools. When a model fails on certain types of inputs, understanding its decision-making process helps identify the root cause. Is the model relying on spurious correlations? Has it learned dataset biases? Are certain features being misinterpreted? Explainability methods allow practitioners to diagnose these issues and refine their models accordingly. The famous case of the “wolf detector” that was actually detecting snow backgrounds Ribeiro, Singh, and Guestrin (2016) illustrates how explanations can reveal that models learn unexpected patterns.

Fairness and bias detection represent another crucial application of explainability. When models exhibit disparate performance across demographic groups, understanding which features drive predictions helps identify sources of bias. If a hiring model heavily weights features correlated with protected attributes, explanations can surface these problematic dependencies, enabling corrective action through feature engineering, reweighting, or architectural changes.

Scientific discovery increasingly relies on machine learning models that identify patterns humans might miss. In drug discovery, models that predict molecular properties need to be interpretable to generate scientific insights about structure-activity relationships Jiménez-Luna et al. (2020). Climate models, protein folding predictions Jumper et al. (2021), and materials science applications all benefit from understanding not just what the model predicts, but why it makes those predictions, as these explanations can suggest new hypotheses and research directions.

16.11.2 The Paradox of Understanding and Utility

The insistence on full explanatory understanding before deploying useful tools, however, represents a historically unusual stance. Many of our most valuable scientific and medical technologies were adopted and saved lives long before we fully understood their mechanisms.

Consider mammography, one of the most important breast cancer screening tools developed in the 20th century. Mammograms began widespread use

in the 1960s and 1970s, and large-scale trials in the 1970s and 1980s demonstrated clear mortality reduction in screened populations Shapiro (1988) Tabar et al. (1985). Yet our understanding of breast cancer biology, tumor heterogeneity, and the precise mechanisms by which early detection improves outcomes continued to evolve for decades afterward. We still grapple with questions about optimal screening intervals, the balance of benefits and harms, and which tumors are truly aggressive versus indolent. The initial adoption was based on empirical evidence of benefit, not complete mechanistic understanding. The lack of complete explanation did not prevent mammography from saving countless lives.

Similarly, many pharmaceutical interventions were discovered through empirical observation long before their mechanisms were understood. Aspirin was used for decades before we understood its inhibition of cyclooxygenase enzymes. Lithium became a treatment for bipolar disorder in the 1940s, but its precise mechanism of action remains incompletely understood. Anesthesia was used successfully for over 150 years before we developed adequate theories of how it works. In each case, careful empirical validation preceded mechanistic understanding, and the lack of explanation did not preclude tremendous benefit.

Engineering provides equally striking examples. The Navier-Stokes equations, which describe fluid flow and form the foundation of aerodynamics, weather prediction, and countless other applications, were formulated in the 19th century. We use numerical solutions to these equations daily for critical applications: designing aircraft, predicting hurricanes, optimizing turbine efficiency, and modeling blood flow in cardiovascular research. Yet one of the Clay Mathematics Institute's Millennium Prize Problems, offering a million-dollar prize, asks whether solutions to the Navier-Stokes equations even exist and are unique in three dimensions Fefferman (2006). We have built an entire technological civilization on equations whose mathematical foundations remain unproven. Engineers successfully use these equations not through complete mathematical understanding but through empirical validation, computational approximation, and careful attention to when the models work well and when they fail.

This historical pattern suggests a more nuanced view of explainability in machine learning. The demand that we fully understand and explain every prediction from a neural network before deploying it represents a higher standard than we have applied to many successful technologies. This is not to argue against explainability research—it is valuable and important. Rather, it suggests that we should focus on:

Empirical validation through rigorous testing, cross-validation, and real-world performance monitoring. A model that consistently makes accurate predictions on held-out test sets and in production may be deployable even if we cannot fully explain every decision.

Understanding failure modes rather than complete mechanistic understanding. Knowing when and where a model is likely to fail, what types of inputs it handles poorly, and how confident it is in its predictions may be more actionable than detailed explanations of every prediction.

Complementary human oversight in high-stakes decisions. Rather than requiring complete explainability, we might deploy powerful but less interpretable models in systems where humans remain in the loop, using model predictions as one input among many.

Appropriate deployment contexts that match interpretability requirements to stakes and alternatives. A model that routes customer service calls can reasonably be less interpretable than one making parole decisions. The comparison should be to existing alternatives: is the model more or less fair, accurate, and interpretable than current practice?

The goal is not to abandon explainability but to maintain perspective. History suggests that empirically validated tools often precede complete understanding, and this gap need not prevent their careful deployment. As we develop more sophisticated explainability methods, we should simultaneously develop more sophisticated frameworks for when and why explanation is necessary, recognizing that the relationship between understanding and utility is subtle and context-dependent.

This balanced view—pursuing explainability while acknowledging the historical precedent for useful tools that outpace understanding—may lead to more productive deployment of machine learning in domains where it can provide real benefit, accompanied by the monitoring, validation, and human oversight appropriate to the stakes involved.

16.12 Model Elaboration and Nested Model Testing

An *elaborated model* in Bayesian statistics refers to a model that extends or generalizes a simpler, baseline (or “underlying”) model by introducing additional parameters or structure. The purpose of elaboration is to capture more complex features of the data, account for possible deviations from the assumptions of the simpler model, or to allow for greater flexibility in modeling.

Formally, suppose we start with a baseline model $f(y | \theta)$, where θ is a parameter of interest. An elaborated model introduces an additional parameter (or set of parameters) λ , resulting in a family of models $f(y | \theta, \lambda)$ indexed by $\lambda \in \Lambda$. The original model is recovered as a special case for some fixed value λ_0 (i.e., $f(y | \theta) = f(y | \theta, \lambda_0)$). The set Λ describes the ways in which the model can be elaborated.

When we use elaborated models then we need to compare nested models (where the simpler model is a special case of the more complex one). In Bayesian analysis, inference in an elaborated model involves integrating over the additional parameters, reflecting uncertainty about both the original and the elaborating parameters.

Applying the usual Bayesian paradigm (disciplined probability accounting) to the elaborated framework, we see that inference about θ is determined by

$$p(\theta | y) = \int_{\Lambda} p(\theta | \lambda, y) p(\lambda | y) d\lambda$$

where

$$\begin{aligned} p(\theta | \lambda, y) &\propto p(y | \theta, \lambda) p(\theta | \lambda) \\ p(\lambda | y) &\propto p(y | \lambda) p(\lambda) \end{aligned}$$

$$\text{where } p(y | \lambda) = \int p(y | \theta, \lambda) p(\theta | \lambda) d\theta$$

For consistency with the elaborated and underlying model, we take $p(\theta | \lambda_0) = p(\theta)$. Since λ labels the form of departure from the initial model $M_0 : \lambda = \lambda_0$, the form of $p(\lambda)$ should be chosen to reflect this departure.

A classical example is the exponential power elaboration of the traditional normal family, allowing for robustness. Here $\lambda \in (0, 3)$ indexes the power and $\lambda_0 = 2$ is the Normal case.

The posterior mean is simply a weighted average with respect to $p(\lambda | y)$:

$$E(\theta | y) = \int E(\theta | \lambda, y) p(\lambda | y) d\lambda = E_{\lambda|y}(E(\theta | \lambda, y))$$

16.12.1 The Dickey-Savage Approach to Nested Models

The Dickey-Savage approach provides a principled Bayesian method for testing nested models—situations where a simpler model is a special case of a more complex one. This approach is particularly useful when we want to assess whether the data support the inclusion of additional parameters or structure in our model, or whether the simpler, baseline model suffices.

In the context of Bayesian hypothesis testing, the Dickey-Savage method allows us to compute the Bayes factor for comparing a nested (elaborated) model to its simpler counterpart using only the posterior and prior distributions of the parameter(s) that distinguish the two models. This not only streamlines the computation but also clarifies the relationship between the models and the evidence provided by the data.

The Bayes Folklore

In essence, the Dickey-Savage result supports the Bayesian strategy of fitting the most comprehensive model possible—sometimes referred to as the ‘elephant’ model—and testing simpler sub-models as special cases. You fit a model M as complex as allowed by your computational budget and statistical skills. The calculations for any nested model, e.g., M_0 , can then be performed entirely within the framework of M .

Suppose you are conducting a hypothesis test comparing two models, M_0 and M_1 . The null hypothesis H_0 is that the simpler model M_0 is true, and the alternative hypothesis H_1 is that the more complex model M_1 is true.

Let’s explore how this approach works and why it is both elegant and practical for model comparison in Bayesian analysis.

Suppose that $M_0 \subset M$. Let (θ, ψ) have matching priors such that

$$p(\psi | \theta = 0, M) = p(\psi | M_0)$$

where $\theta = 0$ corresponds to M_0 . That is, $p(y | \theta = 0, \psi, M) = p(y | \psi, M_0)$.

Then, we can calculate *solely under model M* the Bayes factor as follows:

$$BF = \frac{p(\theta = 0 | y, M)}{p(\theta = 0 | M)} = \frac{p(y | M_0)}{p(y | M)}$$

This is a ratio of posterior ordinates, valid as long as models are nested. By definition of marginals:

$$\begin{aligned} p(y | \theta = 0, M) &= \int p(y | \theta = 0, \psi, M)p(\psi | \theta = 0, M)d\psi \\ &= \int p(y | \psi, M_0)p(\psi | M_0)d\psi \\ &= p(y | M_0) \end{aligned}$$

This elegant result shows that Bayes factors for nested models can be computed entirely within the larger model framework.

16.13 Conclusion

Model selection lies at the heart of the machine learning workflow. Throughout this chapter, we have explored the fundamental tensions that define this task:

the bias-variance trade-off, the pursuit of out-of-sample generalization, and the balance between predictive power and interpretability.

Several key principles emerge. First, in-sample performance is a poor guide to model quality; rigorous out-of-sample evaluation via cross-validation or held-out test sets is essential. Second, the Bayesian framework provides a principled, probabilistic approach to model comparison through the *evidence* or marginal likelihood, which naturally implements Occam’s razor by penalizing unnecessary complexity. Information criteria like BIC offer practical approximations to this ideal. Third, there is no single “best” model for all purposes—a model optimized for prediction may be entirely unsuitable for causal interpretation, and vice versa.

The chapter also engaged with the critical role of explainability. In high-stakes domains, understanding *why* a model makes its predictions is as important as the predictions themselves. We surveyed the modern toolkit of interpretability methods—SHAP, LIME, PDPs, and others—while cautioning against an unrealistic demand for complete mechanistic understanding before deployment. History teaches us that useful tools often precede full explanation.

Ultimately, model selection is an exercise in informed judgment!



17

Statistical Learning Theory and Regularization

“In God we trust; all others must bring data.” — W. Edwards Deming

When AlphaGo defeated world champion Lee Sedol in 2016, its neural networks contained millions of parameters—far more than the number of training positions it had observed. How did it avoid simply memorizing the training data? The answer lies in regularization: the systematic imposition of constraints that prevent models from over-fitting to noise. This chapter reveals that regularization is not merely a computational trick but emerges naturally from Bayesian reasoning about uncertainty.

The development of learning algorithms has been driven by two fundamental paradigms: the classical frequentist approach centered around maximum likelihood estimation (MLE) and the Bayesian approach grounded in decision theory. This chapter explores how these seemingly distinct methodologies converge in modern AI theory, particularly through the lens of regularization and model selection.

Maximum likelihood estimation represents the cornerstone of classical statistical inference. Given observed data $\mathcal{D} = (x_i, y_i)_{i=1}^n$ and a parametric model $f_\theta(x)$, the MLE principle seeks to find the parameter values that maximize the likelihood function:

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta} L(\theta; \mathcal{D}) = \arg \max_{\theta} \prod_{i=1}^n p(y_i | x_i, \theta)$$

This approach has several appealing properties: it provides consistent estimators under mild conditions, achieves the Cramer-Rao lower bound asymptotically, and offers a principled framework for parameter estimation. These classical guarantees are typically stated under i.i.d. sampling assumptions; from a Bayesian perspective, exchangeability (Chapter 3) is a weaker condition that still supports learning from data in many settings. The Cramer-Rao bound is expressed in terms of Fisher information, and in exponential-family models Fisher information and sufficiency are closely related ways of formalizing how much information the data carry about a parameter.

However, MLE has well-documented limitations, particularly in high-dimensional settings. MLE can lead to overfitting, poor generalization, and numerical instability. Furthermore, as shown by Stein's paradox, MLE can be inadmissible, meaning there are other estimators that have lower risk than the MLE. We will start this chapter with the normal means problem and demonstrate how MLE can be inadmissible.

17.1 Normal Means Problem

Consider the vector of means case where $\theta = (\theta_1, \dots, \theta_p)$. We have

$$\bar{y}_i | \theta_i \sim N(\theta_i, \sigma^2/n_i), \quad i = 1, \dots, p, \quad p > 2 \quad (17.1)$$

Here \bar{y}_i is the mean of n_i observations, i.e., $\bar{y}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} y_{ij}$.

The goal is to estimate the vector of means $\theta = (\theta_1, \dots, \theta_p)$, and we can achieve this by borrowing strength across the observations.

This is also a proxy for non-parametric regression, where $y_i = f(x_i) + \varepsilon_i$ and $\theta_i = f(x_i)$. Much has been written on the properties of the Bayes risk as a function of n and p , and extensive work has been done on the asymptotic properties of the Bayes risk as n and p grow to infinity.

The classical inference is based on the CLT

$$\hat{\theta}_i | \theta_i \sim N(\theta_i, \sigma^2/n_i),$$

and the MLE estimate is given by $\hat{\theta}_i = \bar{y}_i$. The MLE estimate is consistent and asymptotically normal, i.e.,

$$\hat{\theta}_i \rightarrow N(\theta_i, \sigma^2/n_i) \quad \text{as } n_i \rightarrow \infty.$$

On the other hand, the Bayes estimator is based on $\theta_i | \hat{\theta}_i$, where $\hat{\theta}_i = \bar{y}_i$. In other words, the classical approach is subject to the *prosecutor's fallacy*—the logical error of confusing $P(\text{data} | \text{hypothesis})$ with $P(\text{hypothesis} | \text{data})$. Classical estimators are unbiased, whereas a Bayes estimator is biased.

Example 17.1 (Screening Corporate Performance). To illustrate the practical importance of the normal means problem, consider the challenge faced by investment analysts screening thousands of publicly traded companies for sustained superior performance (Nicholas G. Polson and Scott 2012). Suppose we have return on assets (ROA) data for $p = 53,038$ firms across 93 countries over 45 years. For each firm i , we observe an average ROA performance \bar{y}_i over some time period.

The fundamental question is: *Which firms have genuinely superior performance versus those that appear successful due to luck?* This is precisely a massive multiple testing problem in the normal means framework. Using MLE, we would estimate each firm's performance as $\hat{\theta}_i = \bar{y}_i$ and declare any firm with $\bar{y}_i > 0$ as superior. However, this approach ignores the massive multiplicity problem—with over 50,000 firms, many will appear successful purely by chance.

The Bayesian approach with appropriate shrinkage priors can distinguish between truly superior firms and those that are merely lucky. As we'll see, this requires heavy-tailed priors that can accommodate the rare firms with genuine outperformance while shrinking the estimates of mediocre firms toward zero. This example demonstrates why the choice between MLE and Bayesian estimation has profound practical consequences in high-dimensional settings.

In practice, the normal means problem often requires careful preprocessing of the observed data. A common and crucial step is to normalize the observations by converting them to z-scores. This standardization serves multiple purposes and connects directly to the theoretical framework we've established.

Consider our corporate performance example where we observe ROA values \bar{y}_i for different firms. These firms may operate in different industries, countries, or time periods, making direct comparison problematic. Raw ROA values might range from -20% to +30%, with different scales and baseline expectations across sectors.

The z-score transformation standardizes each observation relative to a reference distribution:

$$z_i = \frac{\bar{y}_i - \mu_i}{\sigma_i}.$$

where μ_i and σ_i represent the mean and standard deviation estimated and predicted by an autoregressive model.

In the Polson & Scott study (Nicholas G. Polson and Scott 2012), standardization was essential for handling the massive heterogeneity across their dataset of 53,038 firms spanning 93 countries and 45 years. The authors faced three critical challenges that z-score normalization helped address:

- 1. Cross-Country Comparisons:** Raw ROA values varied dramatically across countries due to different accounting standards, economic conditions, and regulatory environments. A 5% ROA in Japan during the 1990s had very different implications than 5% ROA in Brazil during the same period.
- 2. Temporal Adjustments:** Economic cycles, inflation rates, and market conditions changed substantially over the 45-year study period. The authors needed to adjust for these time-varying factors to identify firms with genuinely superior performance rather than those that simply operated during favorable periods.

3. Industry Heterogeneity: Different industries have fundamentally different ROA distributions. Technology firms typically show higher volatility and different baseline performance compared to utilities or manufacturing companies.

The authors implemented a sophisticated normalization procedure:

$$z_{i,t} = \frac{\text{ROA}_{i,t} - \mu_{\text{peer}(i),t}}{\sigma_{\text{peer}(i),t}}$$

where $\text{peer}(i)$ represents firm i 's reference group (defined by industry, country, and size), and the subscript t indicates time-varying adjustments. This created standardized performance measures where:

- $z_{i,t} = 0$ indicates performance exactly at the peer group median
- $z_{i,t} = 1$ indicates performance one standard deviation above peers
- $z_{i,t} = -1$ indicates performance one standard deviation below peers

After this rigorous standardization, the authors discovered a *critical finding*: sustained superior performance ($\theta_i > 0$ consistently over time) was remarkably rare. Most firms showing high raw ROA were simply benefiting from favorable conditions rather than demonstrating genuine operational excellence. This finding emerged only after proper normalization—without standardization, hundreds of firms would have been incorrectly classified as superior performers.

The goal is to estimate the vector θ using squared loss:

$$\mathcal{L}(\theta, \hat{\theta}) = \sum_{i=1}^p (\theta_i - \hat{\theta}_i)^2,$$

where $\hat{\theta}$ is the vector of estimates. We will compare the MLE estimate with the James-Stein estimate. A principled way to evaluate the performance of an estimator is to average its loss over the data; this metric is called the risk. The MLE estimate $\hat{\theta}_i = y_i$ has a constant risk $p\sigma^2$:

$$R(\theta, \hat{\theta}) = \mathbb{E}_y \left(\mathcal{L}(\theta, \hat{\theta}) \right) = \sum_{i=1}^p \mathbb{E}_{y_i} ((y_i - \theta_i)^2).$$

Here the expectation is over the data given by distribution Equation 17.1 and $y_i \sim N(\theta_i, \sigma^2)$, we have $\mathbb{E}_{y_i} ((\theta_i - y_i)^2) = \text{Var}(y_i) = \sigma^2$ for each i . Therefore:

$$R(\theta, \hat{\theta}) = \sum_{i=1}^p \sigma^2 = p\sigma^2.$$

This shows that the MLE risk is constant and does not depend on the true parameter values θ , only on the dimension p and the noise variance σ^2 .

Given that MLE provides a natural baseline estimator with known risk properties, one might ask: can we do better? The Bayesian paradigm offers a fundamentally different perspective that often yields estimators with uniformly lower risk.

Bayesian inference offers a fundamentally different perspective by incorporating prior knowledge and quantifying uncertainty through probability distributions. The Bayesian approach begins with a prior distribution $p(\theta)$ over the parameter space and updates this belief using Bayes' rule:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$$

The *Bayes estimator* is the value $\hat{\theta}^B$ that minimizes the Bayes risk, the expected loss:

$$\hat{\theta}^B = \arg \min_{\hat{\theta}(y)} R(\pi, \hat{\theta}(y))$$

Here π is the prior distribution of θ and $R(\pi, \hat{\theta}(y))$ is the *Bayes risk* defined as:

$$R(\pi, \hat{\theta}(y)) = \mathbb{E}_{\theta \sim \pi} [\mathbb{E}_{y|\theta} [\mathcal{L}(\theta, \hat{\theta}(y))]]. \quad (17.2)$$

For squared error loss, this yields the posterior mean $E(\theta | y)$, while for absolute error loss, it gives the posterior median.

For the normal means problem with squared error loss, this becomes:

$$R(\pi, \hat{\theta}(y)) = \int_{\theta \in \Theta} \left(\int_{y \in Y} (\theta - \hat{\theta}(y))^2 p(y|\theta) dy \right) \pi(\theta) d\theta$$

The Bayes risk quantifies the expected performance of an estimator, taking into account both the uncertainty in the data and the prior uncertainty about the parameter. It serves as a benchmark for comparing different estimators: an estimator with lower Bayes risk is preferred under the chosen prior and loss function. In particular, the Bayes estimator achieves the minimum possible Bayes risk for the given prior and loss.

In 1961, Charles Stein and Willard James proved a startling result: they constructed an estimator for the mean of a multivariate normal distribution that uniformly dominates the sample mean under squared error loss—a finding that challenged what seemed like an elementary law of statistical theory. This result, now known as *Stein's paradox*, demonstrates that for dimensions $p \geq 3$, there always exists an estimator with strictly lower risk than the MLE for all parameter values.

The empirical evidence is striking: in applications ranging from baseball batting averages to toxoplasmosis prevalence rates to Pearson's chi-square tests, the James-Stein estimator achieves mean squared errors less than half that of the sample mean. This result is paradoxical because it overturns the intuition that unbiased estimators should be optimal—despite introducing bias, the James-Stein estimator achieves strictly lower risk for all parameter values when $p \geq 3$. The philosophical implications extend beyond estimation theory: by demonstrating that individual estimates can be improved by considering them jointly, Stein's paradox provides deep connections to Bayesian thinking and the empirical Bayes framework.

Stein's phenomenon where $y_i | \theta_i \sim N(\theta_i, 1)$ and $\theta_i \sim N(0, \tau^2)$ where $\tau \rightarrow \infty$ illustrates this point well. The MLE approach is equivalent to the use of the improper “non-informative” uniform prior and leads to an estimator with poor risk properties.

Let $\|y\|^2 = \sum_{i=1}^p y_i^2$ denote the squared Euclidean norm of y . Then, we can make the following probabilistic statements from the model:

$$P(\|\theta\| > \|y\|) > \frac{1}{2}$$

Now for the posterior, this inequality is reversed under a flat Lebesgue measure:

$$P(\|\theta\| > \|y\| | y) > \frac{1}{2}$$

which is in conflict with the classical statement. This is a property of the prior which leads to a poor rule (the overall average) and risk.

The shrinkage rule (a.k.a. normal prior) where τ^2 is “estimated” from the data avoids this conflict. More precisely, we have:

$$\hat{\theta}(y) = \left(1 - \frac{k-2}{\|y\|^2}\right)y \quad \text{and} \quad E(\|\hat{\theta} - \theta\|^2) < k, \quad \forall \theta.$$

Hence, when $\|y\|^2$ is small, the shrinkage factor is more extreme. For example, if $k = 10$, $\|y\|^2 = 12$, then $\hat{\theta} = (1/3)y$. Now we have the more intuitive result that:

$$P(\|\theta\| > \|y\| | y) < \frac{1}{2}.$$

This shows that careful specification of default priors matters in high dimensions.

The resulting estimator is called the James-Stein estimator and is a shrinkage estimator that shrinks the MLE towards the prior mean. The prior mean is typically the sample mean of the data. The James-Stein estimator is given by:

$$\hat{\theta}_i^{JS} = (1 - \lambda)\hat{\theta}_i^{MLE} + \lambda\bar{y},$$

where λ is a shrinkage parameter and \bar{y} is the sample mean of the data. The shrinkage parameter is typically chosen to minimize the risk of the estimator.

The key idea behind James-Stein shrinkage is that one can “borrow strength” across components. In this sense, the multivariate parameter estimation problem is easier than the univariate one.

Following Efron and Morris (1975), we can view the James-Stein estimator through the lens of empirical Bayes methodology. Efron and Morris demonstrate that Stein’s seemingly paradoxical result has a natural interpretation when viewed as an empirical Bayes procedure that estimates the prior distribution from the data itself.

Consider the hierarchical model:

$$\begin{aligned} y_i | \theta_i &\sim N(\theta_i, \sigma^2) \\ \theta_i | \mu, \tau^2 &\sim N(\mu, \tau^2) \end{aligned}$$

The marginal distribution of y_i is then $y_i \sim N(\mu, \sigma^2 + \tau^2)$. In the empirical Bayes approach, we estimate the hyperparameters μ and τ^2 from the marginal likelihood:

$$m(y | \mu, \tau^2) = \prod_{i=1}^p \frac{1}{\sqrt{2\pi(\sigma^2 + \tau^2)}} \exp\left(-\frac{(y_i - \mu)^2}{2(\sigma^2 + \tau^2)}\right)$$

The maximum marginal likelihood estimators are:

$$\begin{aligned} \hat{\mu} &= \bar{y} = \frac{1}{p} \sum_{i=1}^p y_i \\ \hat{\tau}^2 &= \max\left(0, \frac{1}{p} \sum_{i=1}^p (y_i - \bar{y})^2 - \sigma^2\right) \end{aligned}$$

The empirical Bayes estimator then becomes:

$$\hat{\theta}_i^{EB} = \frac{\hat{\tau}^2}{\sigma^2 + \hat{\tau}^2} y_i + \frac{\sigma^2}{\sigma^2 + \hat{\tau}^2} \hat{\mu}$$

This can be rewritten as:

$$\hat{\theta}_i^{EB} = \left(1 - \frac{\sigma^2}{\sigma^2 + \hat{\tau}^2}\right) y_i + \frac{\sigma^2}{\sigma^2 + \hat{\tau}^2} \bar{y}$$

When $\mu = 0$ and using the estimate $\hat{\tau}^2 = \max(0, \|y\|^2/p - \sigma^2)$, this reduces to a form closely related to the James-Stein estimator:

$$\hat{\theta}_i^{JS} = \left(1 - \frac{(p-2)\sigma^2}{\|y\|^2}\right) y_i$$

Efron and Morris show that the empirical Bayes interpretation provides insight into why the James-Stein estimator dominates the MLE. The key insight is that the MLE implicitly assumes an improper flat prior $\pi(\theta) \propto 1$, which leads to poor risk properties in high dimensions.

The Bayes risk of the James-Stein estimator can be explicitly calculated due to the conjugacy of the normal prior and likelihood:

$$R(\theta, \hat{\theta}^{JS}) = p\sigma^2 - (p-2)\sigma^2 \mathbb{E} \left[\frac{1}{\|\theta + \epsilon\|^2/\sigma^2} \right]$$

where $\epsilon \sim N(0, \sigma^2 I)$. Since the second term is always positive, we have:

$$R(\theta, \hat{\theta}^{JS}) < R(\theta, \hat{\theta}^{MLE}) \quad \forall \theta \in \mathbb{R}^p, \quad p \geq 3$$

This uniform domination demonstrates the **inadmissibility** of the MLE under squared error loss for $p \geq 3$.

In an applied problem, the gap in risk between MLE and JS estimators can be large. For example, in the normal means problem with $p = 100$ and $n = 100$, the risk of the MLE is $R(\theta, \hat{\theta}_{MLE}) = 100$ while the risk of the JS estimator is $R(\theta, \hat{\theta}_{JS}) = 1.5$. The JS estimator is 67 times more efficient than the MLE. The JS estimator is also minimax optimal in the sense that it attains the minimax risk bound for the normal means problem. The minimax risk bound is the smallest risk that can be attained by any estimator.

The James-Stein estimator illustrates how incorporating prior information (via shrinkage) can lead to estimators with lower overall risk compared to the MLE, especially in high-dimensional settings. However, it is not the only shrinkage estimator that dominates the MLE. Other shrinkage estimators, such as the ridge regression estimator, also have lower risk than the MLE. The key insight is that shrinkage estimators can leverage prior information to improve estimation accuracy, especially in high-dimensional settings.

Note, that we used the empirical Bayes version of the definition of risk. Full Bayes approach incorporates both the data and the prior distribution of the parameter as in Equation 17.2.

17.2 Unbiasedness and the Optimality of Bayes Rules

The optimal Bayes rule $\delta_\pi(x) = \mathbb{E}(\theta | y)$ is the posterior mean under squared error loss. An interesting feature of the Bayes rule is that it is biased except

in degenerate cases like improper priors, which can lose optimality properties. This can be seen using the following decomposition.

If the rule was unbiased, then the Bayes risk would be zero. This follows by contradiction: assume for the sake of argument that $E_{x|\theta}(\delta_\pi(y)) = \theta$, then

$$r(\pi, \delta_\pi(y)) = E_\pi \left(E_{y|\theta} ((\delta_\pi(y) - \theta)^2) \right) = E_\pi (\theta^2) + E_y (\delta_\pi(y))^2 - 2E_\pi (\theta E_{y|\theta} (\delta_\pi(y))) = 0$$

which is a contradiction since the Bayes risk cannot be zero in non-degenerate cases.

The key feature of Bayes rules is the bias-variance tradeoff inherent in their nature. You achieve a large reduction in variance for a small amount of bias. This is the underpinning of James-Stein estimation.

Another interesting feature is that the Bayes rule $E(\theta | y)$ is always Bayes sufficient in the sense that

$$E_\pi (\theta | E_\pi (\theta | y)) = E_\pi (\theta | y)$$

So conditioning on $E_\pi (\theta | y)$ is equivalent to conditioning on y when estimating θ . This property is used in quantile neural network approaches to generative methods.

Example 17.2 (Example: James-Stein for Baseball Batting Averages). We reproduce the baseball batting average example from Efron and Morris (1977). Data below has the number of hits for 18 baseball players after 45 at-bats in 1970 season.

Now, we can estimate overall mean and variance

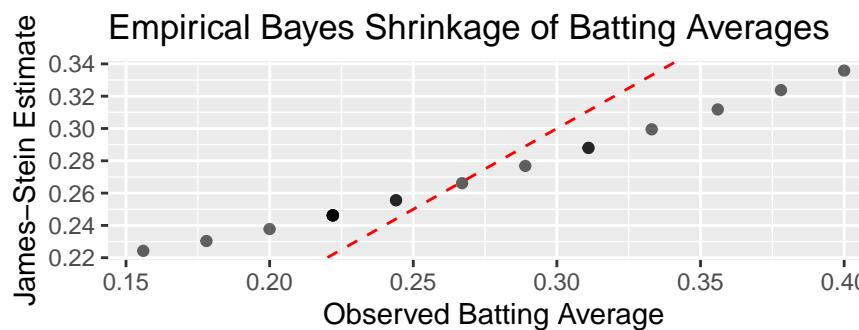
```
mu_hat <- mean(baseball$BattingAverage)
sigma2_hat <- var(baseball$BattingAverage)
```

As well as the posterior mean for each player (James-Stein estimator)

LastName	AtBats	BattingAverage	SeasonAverage	JS
Clemente	45	0.40	0.35	0.34
Robinson	45	0.38	0.31	0.32
Howard	45	0.36	0.28	0.31
Johnstone	45	0.33	0.24	0.30
Berry	45	0.31	0.28	0.29
Spencer	45	0.31	0.27	0.29
Kessinger	45	0.29	0.27	0.28
Alvarado	45	0.27	0.22	0.27
Santo	45	0.24	0.27	0.26

LastName	AtBats	BattingAverage	SeasonAverage	JS
Swaboda	45	0.24	0.23	0.26
Petrocelli	45	0.22	0.26	0.25
Rodriguez	45	0.22	0.22	0.25
Scott	45	0.22	0.30	0.25
Unser	45	0.22	0.26	0.25
Williams	45	0.22	0.25	0.25
Campaneris	45	0.20	0.28	0.24
Munson	45	0.18	0.30	0.23
Alvis	45	0.16	0.18	0.22

Plot below shows the observed averages vs. James-Stein estimate



Calculate mean squared error (MSE) for observed and James-Stein estimates

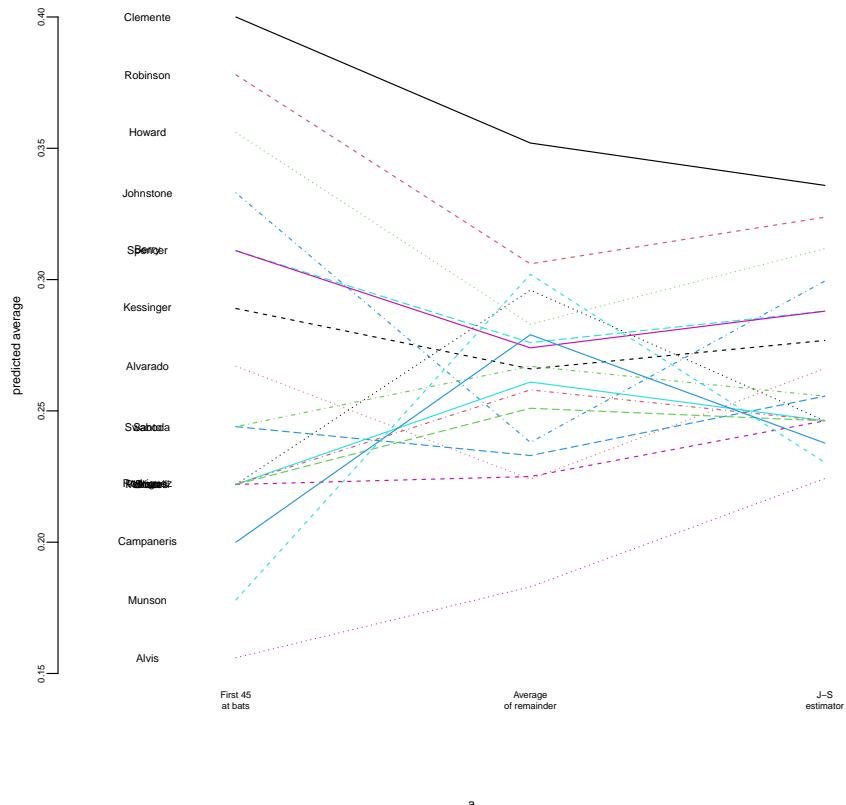
```

mse_observed <- mean((baseball$BattingAverage - mu_hat)^2)
mse_js <- mean((baseball$JS - mu_hat)^2)

cat(sprintf("MSE (Observed): %.6f\n", mse_observed))
## MSE (Observed): 0.004584
cat(sprintf("MSE (James-Stein): %.6f\n", mse_js))
## MSE (James-Stein): 0.001031

```

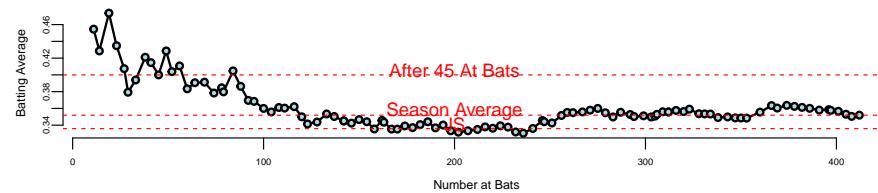
We can see that the James-Stein estimator has a lower MSE than the observed batting averages. This is a demonstration of Stein's paradox, where the James-Stein estimator, which shrinks the estimates towards the overall mean, performs better than the naive sample mean estimator.



Now if we look at the season dynamics for Clemente

```
# Data source:
#<-- https://www.baseball-almanac.com/players/hittinglogs.php?p=clemero01&y=1970
cl <- read.csv("../data/clemente.csv")
x <- cumsum(cl$AB)
y <- cumsum(cl$H) / cumsum(cl$AB)
# Plot x,y starting from index 2
ind <- c(1, 2)
plot(x[-ind], y[-ind], type = "o", ylab = "Batting Average",
      xlab = "Number at Bats", pch = 21, bg = "lightblue", cex =
      0.8, lwd = 2)
# Add horizontal line for season average 145/412 and add text
# above line `Season Average`
text(200, 145 / 412 + 0.005, "Season Average", col = "red")
abline(h = 145 / 412, col = "red", lty = 2)
```

```
# Ted williams record is .406 in 1941, so you know the first
# data points are noise
text(200, baseball$JS[1] + 0.005, "JS", col = "red")
abline(h = baseball$JS[1], col = "red", lty = 2)
text(200, baseball$BattingAverage[1] + 0.005, "After 45 At
# Bats", col = "red")
abline(h = baseball$BattingAverage[1], col = "red", lty = 2)
```



17.2.1 Full Bayes Shrinkage

The alternative approach to the regularization is to use full Bayes, which places a prior distribution on the parameters and computes the *full posterior distribution* using the Bayes rule:

$$p(\theta | \tau, y) = \frac{f(y|\theta)p(\theta | \tau)}{m(y | \tau)},$$

here

$$m(y | \tau) = \int f(y | \theta)p(\theta | \tau)d\theta,$$

Here $m(y | \tau)$ is the marginal beliefs about the data.

The *empirical Bayes* approach is to estimate the prior distribution $p(\theta | \tau)$ from the data. This can be done by maximizing the marginal likelihood $m(y | \tau)$ with respect to τ . The resulting estimator is called the *type II maximum likelihood estimator* (MMLE).

$$\hat{\tau} = \arg \max_{\tau} \log m(y | \tau).$$

For example, in the normal-normal model, when $\theta \sim N(\mu, \tau^2)$ with $\mu = 0$, we can integrate out the high dimensional θ and find $m(y|\tau)$ in closed form as $y_i \sim N(0, \sigma^2 + \tau^2)$

$$m(y|\tau) = (2\pi)^{-n/2}(\sigma^2 + \tau^2)^{-n/2} \exp\left(-\frac{\sum y_i^2}{2(\sigma^2 + \tau^2)}\right)$$

The original JS estimator shrinks to zero and estimates prior variance using empirical Bayes (marginal MLE or Type II MLE). Efron and Morris and Lindley showed that you want to shrink to overall mean \bar{y} and in this approach

$$\theta \sim N(\mu, \tau^2).$$

The original JS is $\mu = 0$. To estimate the μ and τ you can do full Bayes or empirical Bayes that shrinks to overall grand mean \bar{y} , which serves as the estimate of the original prior mean μ . It seems paradoxical that you estimate prior parameters from the data. However, this is not the case. You simply use mixture prior P. Diaconis and Ylvisaker (1983) with marginal MLE (MMLE). The MMLE is the product

$$\int_{\theta_i} \prod_{i=1}^k p(\bar{y}_i | \theta_i) p(\theta_i | \mu, \tau^2).$$

The motivation for the shrinkage prior rather than a flat uniform prior are the following probabilistic arguments. They have an ability to balance signal detection and noise suppression in high-dimensional settings. Unlike flat uniform priors, shrinkage priors adaptively shrink small signals towards zero while preserving large signals. This behavior is crucial for sparse estimation problems, where most parameters are expected to be zero or near-zero. The James-Stein procedure is an example of *global shrinkage*, when the overall sparsity level across all parameters is controlled, ensuring that the majority of parameters are shrunk towards zero. Later in this section we will discuss *local shrinkage* priors, such as the horseshoe prior, which allow individual parameters to escape shrinkage if they represent significant signals.

In summary, flat uniform priors (MLE) fail to provide adequate regularization in high-dimensional settings, leading to poor risk properties and overfitting. By incorporating probabilistic arguments and hierarchical structures, shrinkage priors offer a principled approach to regularization that aligns with Bayesian decision theory and modern statistical practice.

17.3 Bias-Variance Decomposition

The discussion of shrinkage priors and James-Stein estimation naturally leads us to a fundamental concept in statistical learning: the bias-variance decomposition. This decomposition provides the theoretical foundation for understanding why shrinkage methods like James-Stein can outperform maximum likelihood estimation, even when they introduce bias.

The key insight is that estimation error can be decomposed into two components: *bias* (systematic error) and *variance* (random error). While unbiased estimators like maximum likelihood have zero bias, they often suffer from high variance, especially in high-dimensional settings. Shrinkage methods intentionally introduce a small amount of bias to achieve substantial reductions in variance, leading to better overall performance.

This trade-off between bias and variance is not just a theoretical curiosity—it's the driving force behind many successful machine learning algorithms, from ridge regression to neural networks with dropout. Understanding this decomposition helps us make principled decisions about model complexity and regularization.

For parameter estimation, we can decompose the mean squared error as follows:

$$\begin{aligned} E((\hat{\theta} - \theta)^2) &= E((\hat{\theta} - E(\hat{\theta}) + E(\hat{\theta}) - \theta)^2) \\ &= E((\hat{\theta} - E(\hat{\theta}))^2) + E((E(\hat{\theta}) - \theta)^2) \\ &= \text{Var}(\hat{\theta}) + \text{Bias}(\hat{\theta})^2 \end{aligned}$$

The cross term has expectation zero.

For prediction problems, we have $y = \theta + \epsilon$ where ϵ is independent with $\text{Var}(\epsilon) = \sigma^2$. Hence

$$E((y - \hat{y})^2) = \sigma^2 + E((\hat{\theta} - \theta)^2)$$

This decomposition shows that prediction error consists of irreducible noise, estimation variance, and estimation bias. Bayesian methods typically trade a small increase in bias for a substantial reduction in variance, leading to better overall performance.

The bias-variance decomposition provides a framework for understanding estimation error, but it doesn't tell us how to construct optimal estimators. To find the best decision rule—whether for estimation, hypothesis testing, or model selection—we need a more general framework that can handle different types of loss functions and prior information. This leads us to Bayesian decision theory and the concept of risk decomposition.

17.3.1 Risk Decomposition

How does one find an optimal decision rule? It could be a test region, an estimation procedure or the selection of a model. Bayesian decision theory addresses this issue.

The *a posteriori* Bayes risk approach is as follows. Let $\delta(x)$ denote the decision

rule. Given the prior $\pi(\theta)$, we can simply calculate

$$R_n(\pi, \delta) = \int_x m(x) \left\{ \int_{\Theta} \mathcal{L}(\theta, \delta(x)) p(\theta|x) d\theta \right\} dx.$$

Then the optimal Bayes rule is to *pointwise* minimize the inner integral (a.k.a. the posterior Bayes risk), namely

$$\delta^*(x) = \arg \max_{\delta} \int_{\Theta} \mathcal{L}(\theta, \delta(x)) p(\theta|x) d\theta.$$

The caveat is that this gives us no intuition into the characteristics of the prior which are important. Moreover, we do not need the marginal beliefs $m(x)$.

For example, under squared error estimation loss, the optimal estimator is simply the posterior mean, $\delta^*(x) = E(\theta|x)$. The properties of this optimal rule—including its inherent bias and the bias-variance tradeoff it embodies—were established in Section 17.2.

17.4 Sparsity

Let the true parameter be sparse with the form $\theta_p = (\sqrt{d/p}, \dots, \sqrt{d/p}, 0, \dots, 0)$. The problem of recovering a vector with many zero entries is called sparse signal recovery. The “ultra-sparse” or “nearly black” vector case occurs when p_n , denoting the number of non-zero parameter values, satisfies $\theta \in l_0[p_n]$, which denotes the set $\#(\theta_i \neq 0) \leq p_n$ where $p_n = o(n)$ and $p_n \rightarrow \infty$ as $n \rightarrow \infty$.

High-dimensional predictor selection and sparse signal recovery are routine statistical and machine learning tasks and present a challenge for classical statistical methods. Historically, James-Stein estimation (or ℓ_2 -regularization) functions as a global shrinkage rule. Because it lacks local parameters to adapt to sparsity, it struggles to recover sparse signals effectively.

James-Stein is equivalent to the model:

$$y_i = \theta_i + \epsilon_i \quad \text{and} \quad \theta_i \sim \mathcal{N}(0, \tau^2)$$

For the sparse r -spike problem, $\hat{\theta}_{JS}$ performs poorly and we require a different rule. rather than using softmax For θ_p we have:

$$\frac{p\|\theta\|^2}{p + \|\theta\|^2} \leq R(\hat{\theta}^{JS}, \theta_p) \leq 2 + \frac{p\|\theta\|^2}{d + \|\theta\|^2}.$$

This implies that $R(\hat{\theta}^{JS}, \theta_p) \geq (p/2)$.

In the sparse case, a simple thresholding rule can beat MLE and JS when the signal is sparse. Assuming $\sigma^2 = 1$, the thresholding estimator is:

$$\hat{\theta}_{thr} = \begin{cases} \hat{\theta}_i & \text{if } \hat{\theta}_i > \sqrt{2 \ln p} \\ 0 & \text{otherwise} \end{cases}$$

This simple example shows that the choice of penalty should not be taken for granted as different estimators will have different risk profiles.

One such estimator that achieves the optimal minimax rate is the horseshoe estimator Carlos M. Carvalho, Polson, and Scott (2010), which we discuss in detail in Section 17.10. It is a local shrinkage estimator that dominates the sample mean in MSE and has good posterior concentration properties for sparse signal problems.

17.5 Maximum A posteriori Estimation (MAP) and Regularization

Having established that Bayesian shrinkage reduces estimation risk, we now examine a computationally tractable alternative: maximum a posteriori (MAP) estimation. While Bayesian shrinkage provably reduces estimation risk, full posterior inference can be computationally demanding. Maximum a posteriori (MAP) estimation offers a tractable alternative that captures many regularization benefits without the cost of full integration. MAP captures the regularization benefits of Bayesian methods without requiring full posterior inference.

Given input-output pairs (x_i, y_i) , MAP learns the function f that maps inputs x_i to outputs y_i by minimizing

$$\underset{f}{\text{minimize}} \quad \sum_{i=1}^N \mathcal{L}(y_i, f(x_i)) + \lambda \phi(f).$$

The first term is the *loss function* that measures the difference between the predicted output $f(x_i)$ and the true output y_i . The second term is a *regularization term* that penalizes complex functions f to prevent overfitting. The parameter λ controls the trade-off between fitting the data well and keeping the function simple. In the case when f is a parametric model, then we simply replace f with the parameters θ of the model, and the regularization term becomes a penalty on the parameters.

The loss is simply a negative log-likelihood from a probabilistic model specified for the data generating process. For example, when y is numeric and $y_i |$

$x_i \sim N(f(x_i), \sigma^2)$, we get the squared loss $\mathcal{L}(y, f(x)) = (y - f(x))^2$. When $y_i \in \{0, 1\}$ is binary, we use the logistic loss $\mathcal{L}(y, f(x)) = \log(1 + \exp(-yf(x)))$.

The penalty term $\lambda\phi(f)$ discourages complex functions f . Then, we can think of regularization as a technique to incorporate some prior knowledge about parameters of the model into the estimation process. Consider an example when regularization allows us to solve a hard problem of filtering noisy traffic data.

Example 17.3 (Traffic). Consider traffic flow speed measured by an in-ground sensor installed on interstate I-55 near Chicago. Speed measurements are noisy and prone to have outliers. Figure 17.1 shows measured speed data, averaged over five minute intervals on one of the weekdays.

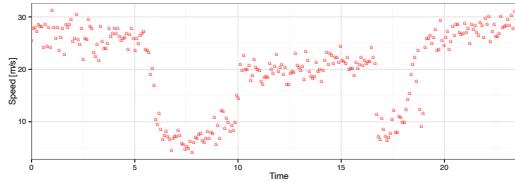


Figure 17.1: Speed profile over 24 hour period on I-55, on October 22, 2013

The statistical model is

$$y_t = f_t + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2), \quad t = 1, \dots, n,$$

where y_t is the speed measurement at time t , f_t is the true underlying speed at time t , and ϵ_t is the measurement noise. There are two sources of noise. The first is the measurement noise, caused by the inherent nature of the sensor's hardware. The second source is due to sampling error, vehicles observed on a specific lane where the sensor is installed might not represent well traffic in other lanes. A naive MLE approach would be to estimate the speed profile $f = (f_1, \dots, f_n)$ by minimizing the squared loss

$$\hat{f} = \arg \min_f \sum_{t=1}^n (y_t - f_t)^2.$$

However, the minima of this loss function is 0 and corresponds to the case when $\hat{f}_t = y_t$ for all t . We have learned nothing about the speed profile, and the estimate is simply the noisy observation y_t . In this case, we have no way to distinguish between the true speed profile and the noise.

However, we can use regularization and bring some prior knowledge about traffic speed profiles to improve the estimate of the speed profile and to remove the noise.

Specifically, we will use a *trend filtering* approach. Under this approach, we assume that the speed profile f is a piece-wise linear function of time, and we want to find a function that captures the underlying trend while ignoring the noise. The regularization term $\phi(f)$ is then the second difference of the speed profile,

$$\lambda \sum_{t=1}^{n-1} |f_{t-1} - 2f_t + f_{t+1}|$$

which penalizes the “kinks” in the speed profile. The value of this penalty is zero, when f_{t-1}, f_t, f_{t+1} lie on a straight line, and it increases when the speed profile has a kink. The parameter λ is a regularization parameter that controls the strength of the penalty.

Trend filtering penalized function is then

$$(1/2) \sum_{t=1}^n (y_t - f_t)^2 + \lambda \sum_{t=1}^{n-1} |f_{t-1} - 2f_t + f_{t+1}|,$$

which is a variation of a well-known Hodrick-Prescott filter.

This approach requires us to choose the regularization parameter λ . A small value of λ will lead to a function that fits the data well, but may not capture the underlying trend. A large value of λ will lead to a function that captures the underlying trend, but may not fit the data well. The optimal value of λ can be chosen using cross-validation or other model selection techniques. The left panel of Figure 17.2 shows the trend filtering for different values of $\lambda \in \{5, 50, 500\}$. The right panel shows the optimal value of λ chosen by cross-validation (by visual inspection).

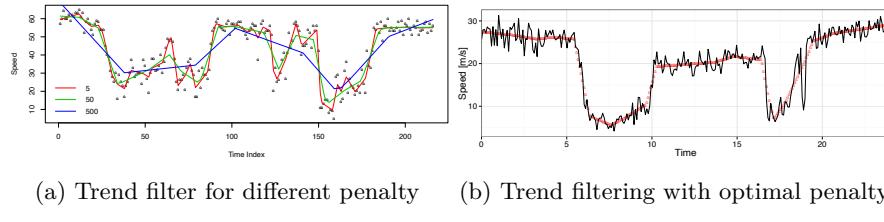


Figure 17.2: Trend Filtering for Traffic Speed Data

17.6 The Duality Between Regularization and Priors

A large number of statistical problems can be expressed in the canonical optimization form:

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad l(x) + \phi(x) \quad (17.3)$$

Perhaps the most common example arises in estimating the regression coefficients x in a generalized linear model. Here $l(x)$ is a negative log likelihood or some other measure of fit, and $\phi(x)$ is a penalty function that effects a favorable bias-variance tradeoff.

From the Bayesian perspective, $l(x)$ and $\phi(x)$ correspond to the negative logarithms of the sampling model and prior distribution in the hierarchical model:

$$p(y | x) \propto \exp\{-l(x)\}, \quad p(x) \propto \exp\{-\phi(x)\} \quad (17.4)$$

and the solution to Equation 17.3 may be interpreted as a maximum *a posteriori* (MAP) estimate.

Another common case is where x is a variable in a decision problem where options are to be compared on the basis of expected loss, and where $l(x)$ and $\phi(x)$ represent conceptually distinct contributions to the loss function. For example, $l(x)$ may be tied to the data, and $\phi(x)$ to the intrinsic cost associated with the decision.

17.6.1 MAP as a Poor Man's Bayesian

There is a duality between using regularization term in optimization problem and assuming a prior distribution over the parameters of the model f . Given the likelihood $L(y_i, f(x_i))$, the posterior is given by Bayes' rule:

$$p(f | y, x) = \frac{\prod_{i=1}^n L(y_i, f(x_i))p(f)}{p(y | x)}.$$

If we take the negative log of this posterior, we get:

$$-\log p(f | y, x) = -\sum_{i=1}^n \log L(y_i, f(x_i)) - \log p(f) + \log p(y | x).$$

Since loss is the negative log-likelihood $\mathcal{L}(y_i, f(x_i)) = -\log L(y_i, f(x_i))$, the posterior maximization is equivalent to minimizing the following regularized loss function:

$$\sum_{i=1}^n \mathcal{L}(y_i, f(x_i)) - \log p(f).$$

The last term $\log p(y_i | x_i)$ does not depend on f and can be ignored in the optimization problem. Thus, the equivalence is given by:

$$\lambda\phi(f) = -\log p(f),$$

where $\phi(f)$ is the penalty term that corresponds to the prior distribution of f . Below we will consider several choices for the prior distribution of f and the corresponding penalty term $\phi(f)$ commonly used in practice.

17.7 Ridge Regression (ℓ_2 Norm)

17.7.1 Tikhonov Regularization Framework

The Tikhonov regularization framework provides a general setting for regression that connects classical regularization theory with modern Bayesian approaches. Given observed data $(x_i, y_i)_{i=1}^n$ and a parametric model $f(x, w)$ with parameter vector $w \in \mathbb{R}^k$, we define a data misfit functional:

$$E_D(w) = \sum_{i=1}^n (y_i - f(x_i, w))^2$$

This yields a Gaussian likelihood:

$$p(y | w) = \frac{1}{Z_D} \exp\left(-\frac{1}{2\sigma^2} E_D(w)\right)$$

where σ^2 denotes the noise variance and Z_D is a normalization constant.

A Gaussian prior on the weights can be specified as:

$$p(w) = \frac{1}{Z_W} \exp\left(-\frac{1}{2\sigma_w^2} E_W(w)\right)$$

where $E_W(w)$ denotes a quadratic penalty on w (e.g., $E_W(w) = w^T w$), σ_w^2 is the prior variance, and Z_W is its normalization constant.

The hyperparameters σ^2 and σ_w^2 control the strength of the noise and the prior. Following MacKay's notation, we define precisions $\tau_D^2 = 1/\sigma^2$ and $\tau_w^2 = 1/\sigma_w^2$. Let B and C denote the Hessians of $E_D(w)$ and $E_W(w)$ at the maximum a posteriori (MAP) estimate w^{MAP} . The Hessian of the negative log-posterior is then $A = \tau_w^2 C + \tau_D^2 B$.

Evaluating the log-evidence under a Gaussian Laplace approximation yields:

$$\log p(D | \tau_w^2, \tau_D^2) = -\tau_w^2 E_W^{\text{MAP}} - \tau_D^2 E_D^{\text{MAP}} - \frac{1}{2} \log \det A - \log Z_W(\tau_w^2) - \log Z_D(\tau_D^2) + \frac{k}{2} \log(2\pi)$$

The associated Occam factor, which penalizes excessively small prior variances, is:

$$-\tau_w^2 E_W^{\text{MAP}} - \frac{1}{2} \log \det A - \log Z_W(\tau_w^2)$$

This represents the reduction in effective volume of parameter space from prior to posterior.

[!NOTE] **Notation:** In the previous sections on Normal Means, we used θ to denote the parameters. In the context of regression, we will follow standard convention and use β to denote the vector of coefficients.

The ridge regression uses a Gaussian prior on the parameters of the model f , which leads to a squared penalty term. Specifically, we assume that the parameters β of the model $f(x) = x^T \beta$ are distributed as:

$$\beta \sim N(0, \sigma_\beta^2 I),$$

where I is the identity matrix and σ_β^2 is the prior variance (distinct from the noise variance σ^2). The prior distribution of β is a multivariate normal distribution with mean 0 and covariance $\sigma_\beta^2 I$. The negative log of this prior distribution is given by:

$$-\log p(\beta) = \frac{1}{2\sigma_\beta^2} \|\beta\|_2^2 + \text{const},$$

where $\|\beta\|_2^2 = \sum_{j=1}^p \beta_j^2$ is the squared 2-norm of the vector β . The regularization term $\phi(f)$ is then given by:

$$\phi(f) = \frac{1}{2\sigma_\beta^2} \|\beta\|_2^2.$$

This leads to the following optimization problem:

$$\underset{\beta}{\text{minimize}} \quad \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2,$$

where $\lambda = 1/\sigma^2$ is the regularization parameter that controls the strength of the prior. The solution to this optimization problem is given by:

$$\hat{\beta}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y.$$

The regularization parameter λ is related to the variance of the prior distribution. When $\lambda = 0$, the function f is the maximum likelihood estimate of the parameters. When λ is large, the function f is the prior mean of the parameters. When λ is infinite, the function f is the prior mode of the parameters.

Notice, that the OLS estimate (invented by Gauss) is a special case of ridge regression when $\lambda = 0$:

$$\hat{\beta}_{\text{OLS}} = (X^T X)^{-1} X^T y.$$

The original motivation for ridge regularization was to address the problem of numerical instability in the OLS solution when the design matrix X is ill-conditioned, i.e. when $X^T X$ is close to singular. In this case, the OLS solution can be very sensitive to small perturbations in the data, leading to large variations in the estimated coefficients $\hat{\beta}$. This is particularly problematic when the number of features p is large, as the condition number of $X^T X$ can grow rapidly with p . The ridge regression solution stabilizes the OLS solution by adding a small positive constant λ to the diagonal of the $X^T X$ matrix, which improves the condition number and makes the solution more robust to noise in the data. The additional term λI simply shifts the eigenvalues of $X^T X$ away from zero, thus improving the numerical stability of the inversion.

Another way to think and write the objective function of Ridge as the following constrained optimization problem:

$$\underset{\beta}{\text{minimize}} \quad \|y - X\beta\|_2^2 \quad \text{subject to} \quad \|\beta\|_2^2 \leq t,$$

where t is a positive constant that controls the size of the coefficients β . This formulation emphasizes the idea that ridge regression is a form of regularization that constrains the size of the coefficients, preventing them from growing too large and leading to overfitting. The constraint $\|\beta\|_2^2 \leq t$ can be interpreted as a budget on the size of the coefficients, where larger values of t allow for larger coefficients and more complex models.

Constraint on the model parameters (and the original Ridge estimator) was proposed by Andrey Nikolayevich Tikhonov et al. (1943) for solving inverse problems to “discover” physical laws from observations. The norm of the β vector would usually represent amount of energy required. Many processes in nature are energy minimizing!

Again, the tuning parameter λ controls trade-off between how well model fits the data and how small β s are. Different values of λ will lead to different models. We select λ using cross validation.

Example 17.4 (Shrinkage). Consider a simulated data with $n = 50$, $p = 30$, and $\sigma^2 = 1$. The true model is linear with 10 large coefficients between 0.5 and 1.

Our approximators \hat{f}_β is a linear regression. We can empirically calculate the bias by calculating the empirical squared loss $1/n\|y - \hat{y}\|_2^2$ and variance can be empirically calculated as $1/n\sum(\hat{y} - \hat{y}_i)^2$

Bias squared $\text{Bias}(\hat{y})^2 = 0.006$ and variance $\text{Var}(\hat{y}) = 0.627$. Thus, the prediction error $= 1 + 0.006 + 0.627 = 1.633$

We'll do better by shrinking the coefficients to reduce the variance. Let's estimate how big a gain we can achieve with Ridge regression.

The figure below shows the distribution of true coefficient values used to generate the data. The histogram reveals a bimodal structure: approximately

20 coefficients are exactly zero (the tall bar at 0.0), while 10 coefficients are non-zero with values concentrated between 0.5 and 1.0. This sparse structure, where most features are irrelevant and only a subset truly matter for prediction, is common in many real-world problems such as genomics, text analysis, and high-dimensional sensor data.

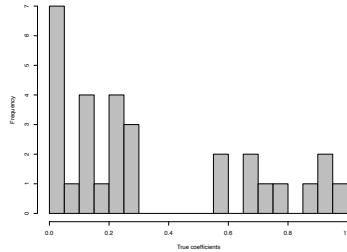


Figure 17.3: True model coefficients

The key question is: what value of the regularization parameter λ should we choose? Too small, and we don't shrink enough to reduce variance; too large, and we introduce excessive bias by shrinking the true non-zero coefficients toward zero. The optimal choice balances these competing concerns.

Figure 17.4a shows how prediction error varies with the amount of shrinkage (controlled by λ). The horizontal dashed line represents the constant prediction error of 1.633 from ordinary linear regression (OLS). The red curve shows Ridge regression's prediction error as we increase regularization. At low shrinkage (left side), Ridge behaves similarly to OLS with high variance. As we increase shrinkage, the prediction error decreases substantially, reaching a minimum around $\lambda \approx 5$. Beyond this point, excessive shrinkage introduces too much bias, and prediction error begins to rise again. The U-shaped curve is characteristic of the bias-variance trade-off: we need enough regularization to stabilize predictions, but not so much that we distort the true signal.

Figure 17.4b decomposes Ridge regression's prediction error into its constituent parts as a function of λ . The black dashed horizontal line at approximately 1.0 represents the irreducible error from noise in the data. The blue curve (Ridge Var) shows variance decreasing monotonically as λ increases: stronger regularization makes the model more stable across different training sets. The red curve (Ridge Bias²) shows squared bias increasing as λ grows: we move further from the true model by shrinking coefficients. The black curve (Ridge MSE) is the sum of these components plus the irreducible error. The optimal λ occurs where the rate of variance reduction exactly balances the rate of bias increase, minimizing total prediction error.

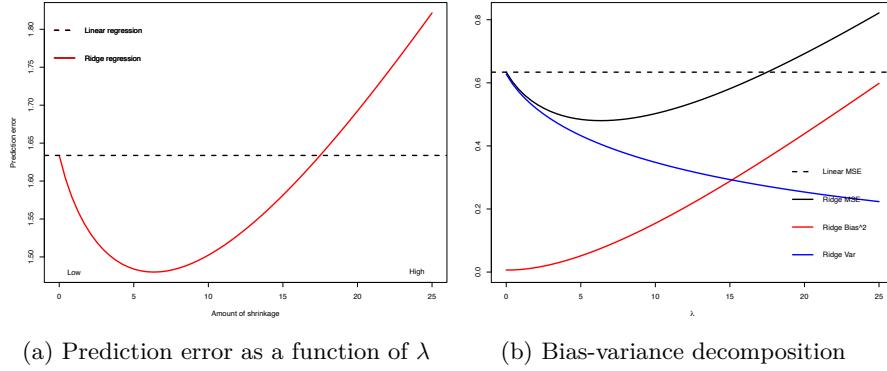


Figure 17.4: Ridge regression performance metrics

At the optimal value of λ , Ridge regression achieves squared bias of 0.077 and variance of 0.402, yielding a total prediction error of $1 + 0.077 + 0.402 = 1.48$. Compare this to the OLS solution with squared bias of 0.006 and variance of 0.627, giving prediction error of $1 + 0.006 + 0.627 = 1.633$. By accepting a modest increase in bias (from 0.006 to 0.077), we achieve a substantial reduction in variance (from 0.627 to 0.402), resulting in an overall improvement of approximately 9% in prediction error.

This example illustrates a fundamental principle in statistical learning: the best predictor is not necessarily the one that fits the training data most closely. By deliberately introducing bias through regularization, we can build models that generalize better to new data. The Bayesian perspective makes this trade-off explicit: the prior distribution encodes our belief that coefficients should be small, and the posterior balances this belief against the evidence in the data.

17.7.2 Kernel View of Ridge Regression

Another interesting view stems from what is called the push-through matrix identity:

$$(aI + UV)^{-1}U = U(aI + VU)^{-1}$$

for a, U, V such that the products are well-defined and the inverses exist. We can obtain this from $U(aI + VU) = (aI + UV)U$, followed by multiplication by $(aI + UV)^{-1}$ on the left and the right. Applying the identity above to the ridge regression solution with $a = \lambda$, $U = X^T$, and $V = X$, we obtain an alternative form for the ridge solution:

$$\hat{\beta} = X^T(XX^T + \lambda I)^{-1}Y.$$

This is often referred to as the kernel form of the ridge estimator. From this, we can see that the ridge fit can be expressed as

$$X\hat{\beta} = XX^T(XX^T + \lambda I)^{-1}Y.$$

What does this remind you of? This is precisely $K(K + \lambda I)^{-1}Y$ where $K = XX^T$, which, recall, is the fit from RKHS regression with a linear kernel $k(x, z) = x^Tz$. Therefore, we can think of RKHS regression as generalizing ridge regression by replacing the standard linear inner product with a general kernel. (Indeed, RKHS regression is often called kernel ridge regression.)

17.8 Scale Mixtures Representations

Why should we care about expressing distributions as mixtures? The answer lies in computational tractability and theoretical insight. Many important priors used in sparse estimation—including the Laplace (Lasso), horseshoe, and logistic—can be represented as Gaussian distributions with random variance. This representation is far more than a mathematical curiosity: it enables efficient Gibbs sampling algorithms where each conditional distribution has a closed form, and it reveals deep connections between seemingly different regularization approaches.

Scale mixtures of normals provide a powerful framework for constructing flexible priors and computational algorithms in Bayesian statistics. The key insight is that many useful distributions can be represented as Gaussian distributions with random variance, leading to tractable MCMC algorithms and analytical insights.

17.9 Lasso Regression (ℓ_1 Norm)

The Lasso (Least Absolute Shrinkage and Selection Operator) regression uses a Laplace prior on the parameters of the model f , which leads to an ℓ_1 penalty term. Specifically, we assume that the parameters β of the model $f(x) = x^T\beta$ are distributed as:

$$\beta_j \sim \text{Laplace}(0, b) \quad \text{independently for } j = 1, \dots, p,$$

where $b > 0$ is the scale parameter. The Laplace distribution has the probability density function:

$$p(\beta_j | b) = \frac{1}{2b} \exp\left(-\frac{|\beta_j|}{b}\right)$$

and is shown in Figure 17.5.

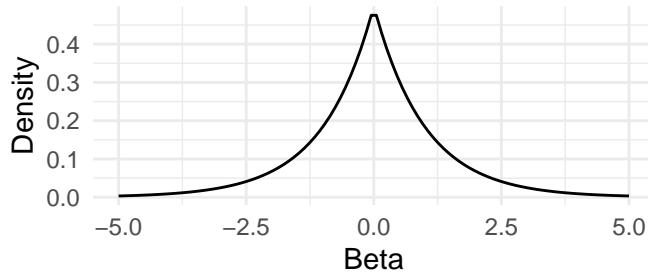


Figure 17.5: Laplace Distribution PDF

The negative log of this prior distribution is given by:

$$-\log p(\beta) = \frac{1}{b} \|\beta\|_1 + \text{const},$$

where $\|\beta\|_1 = \sum_{j=1}^p |\beta_j|$ is the ℓ_1 -norm of the vector β . The regularization term $\phi(f)$ is then given by:

$$\phi(f) = \frac{1}{b} \|\beta\|_1.$$

This leads to the following optimization problem:

$$\underset{\beta}{\text{minimize}} \quad \|y - X\beta\|_2^2 + \lambda \|\beta\|_1,$$

where $\lambda = 2\sigma^2/b$ is the regularization parameter that controls the strength of the prior. Unlike ridge regression, the Lasso optimization problem does not have a closed-form solution due to the non-differentiable nature of the ℓ_1 penalty. However, efficient algorithms such as coordinate descent and proximal gradient methods can be used to solve it.

The key distinguishing feature of Lasso is its ability to perform automatic variable selection. The ℓ_1 penalty encourages sparsity in the coefficient vector $\hat{\beta}$, meaning that many coefficients will be exactly zero. This property makes Lasso particularly useful for high-dimensional problems where feature selection is important.

When $\lambda = 0$, the Lasso reduces to the ordinary least squares (OLS) estimate. As λ increases, more coefficients are driven to exactly zero, resulting in a sparser model. When λ is very large, all coefficients become zero.

The geometric intuition behind Lasso's sparsity-inducing property comes from the constraint formulation. We can write the Lasso problem as:

$$\underset{\beta}{\text{minimize}} \quad \|y - X\beta\|_2^2 \quad \text{subject to} \quad \|\beta\|_1 \leq t,$$

where t is a positive constant that controls the sparsity of the solution. The constraint region $\|\beta\|_1 \leq t$ forms a diamond (in 2D) or rhombus-shaped region with sharp corners at the coordinate axes. The optimal solution often occurs at these corners, where some coefficients are exactly zero.

From a Bayesian perspective, the Lasso estimator corresponds to the maximum a posteriori (MAP) estimate under independent Laplace priors on the coefficients. We use Bayes rule to calculate the posterior as a product of Normal likelihood and Laplace prior:

$$-\log p(\beta | y, b) = \|y - X\beta\|_2^2 + \frac{2\sigma^2}{b} \|\beta\|_1.$$

For fixed σ^2 and $b > 0$, the posterior mode is equivalent to the Lasso estimate with $\lambda = 2\sigma^2/b$. Large variance b of the prior is equivalent to small penalty weight λ in the Lasso objective function.

One of the most popular algorithms for solving the Lasso problem is coordinate descent. The algorithm iteratively updates each coefficient while holding all others fixed. For the j -th coefficient, the update rule is:

$$\hat{\beta}_j \leftarrow \text{soft} \left(\frac{1}{n} \sum_{i=1}^n x_{ij}(y_i - \sum_{k \neq j} x_{ik}\hat{\beta}_k), \frac{\lambda}{n} \right),$$

where the soft-thresholding operator is defined as:

$$\text{soft}(z, \gamma) = \text{sign}(z)(|z| - \gamma)_+ = \begin{cases} z - \gamma & \text{if } z > \gamma \\ 0 & \text{if } |z| \leq \gamma \\ z + \gamma & \text{if } z < -\gamma \end{cases}$$

Example 17.5 (Sparsity and Variable Selection). We will demonstrate the Lasso's ability to perform variable selection and shrinkage using simulated data. The data will consist of a design matrix with correlated predictors and a sparse signal, where only a 5 out of 20 predictors have non-zero coefficients.

```
# True coefficients - sparse signal
beta_true <- c(3, -2, 1.5, 0, 0, 2, 0, 0, 0, -1, rep(0, 10))
sparse_indices <- which(beta_true != 0)
```

```
# Generate response
y <- X %*% beta_true + sigma * rnorm(n)
```

Then we use `glmnet` package to fit the Lasso model and visualize the coefficient paths. We will also perform cross-validation to select the optimal regularization parameter λ .

```
# Fit LASSO path using glmnet
library(glmnet)
lasso_fit <- glmnet(X, y, alpha = 1)
# Plot coefficient paths
plot(lasso_fit, xvar = "lambda", label = TRUE)
```

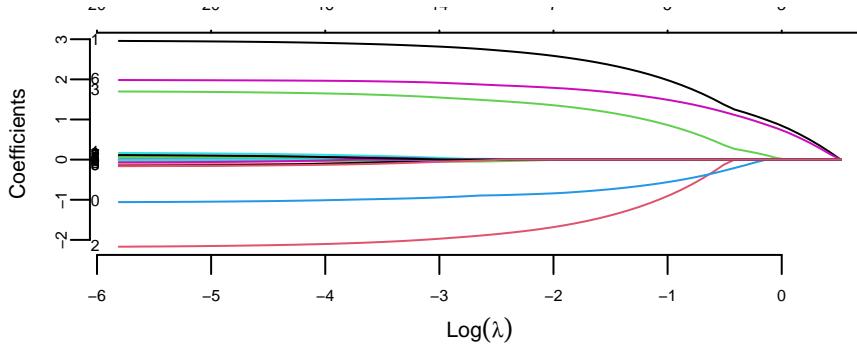


Figure 17.6: LASSO Coefficient Paths

The coefficient paths plot shows how LASSO coefficients shrink toward zero as the regularization parameter lambda increases. The colored lines represent different predictors, demonstrating LASSO's variable selection property. Note, that `glmnet` fitted the model for a sequence of λ values. The algorithms starts with a large lambda value, where all coefficients are penalized to zero. Then, it gradually decreases lambda, using the coefficients from the previous, slightly more penalized model as a “warm start” for the current calculation. This pathwise approach is significantly more efficient than starting the optimization from scratch for every single λ . By default, `glmnet` computes the coefficients for a sequence of 100 lambda values spaced evenly on the logarithmic scale, starting from a data-driven maximum value (where all coefficients are zero) down to a small fraction of that maximum. The user can specify their own sequence of lambda values if specific granularity or range is desired.

Finally, we will perform cross-validation to select the optimal λ value and compare the estimated coefficients with the true values.

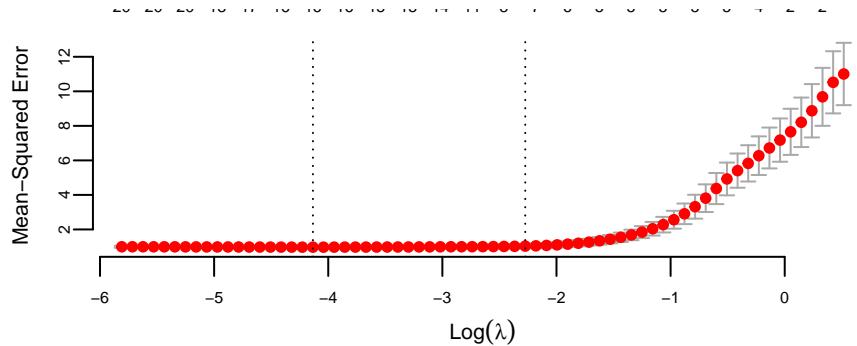


Figure 17.7: Cross-validation for LASSO

Now, we can extract the coefficients `lambda.min` and `lambda.1se` from the cross-validation results, which correspond to the minimum cross-validated error and the most regularized model within one standard error of the minimum, respectively.

```
# Extract coefficients at optimal lambda
lambda_min <- cv_lasso$lambda.min
lambda_1se <- cv_lasso$lambda.1se
coef_min <- coef(lasso_fit, s = lambda_min)
coef_1se <- coef(lasso_fit, s = lambda_1se)
# Print values of lambda
cat("Optimal lambda (min):", lambda_min, "\n")
## Optimal lambda (min): 0.016
cat("Optimal lambda (1se):", lambda_1se, "\n")
## Optimal lambda (1se): 0.1
```

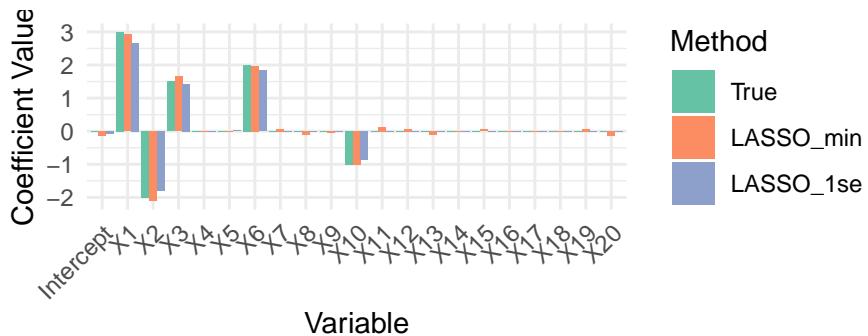


Figure 17.8: Coefficient Estimates Comparison

It seems like LASSO has successfully identified the non-zero coefficients and

shrunk the noise variables to zero. The coefficient estimates at `lambda.min` and `lambda.1se` show that LASSO retains the true signals while effectively ignoring the noise. Let's calculate the prediction errors and evaluate the variable selection performance of LASSO at both optimal λ values.

```
# Calculate prediction errors
pred_min <- predict(lasso_fit, newx = X, s = lambda_min)
pred_1se <- predict(lasso_fit, newx = X, s = lambda_1se)

mse_min <- mean((y - pred_min)^2)
mse_1se <- mean((y - pred_1se)^2)

cat("Mean Squared Error (lambda.min):", round(mse_min, 3), "\n")
## Mean Squared Error (lambda.min): 0.68
cat("Mean Squared Error (lambda.1se):", round(mse_1se, 3), "\n")
## Mean Squared Error (lambda.1se): 0.85
```

In summary, this example demonstrates how LASSO regression can be used for both variable selection and regularization in high-dimensional settings. By tuning the regularization parameter λ , LASSO is able to shrink irrelevant coefficients to zero, effectively identifying the true underlying predictors while controlling model complexity. The comparison of coefficient estimates and prediction errors at different λ values highlights the trade-off between model sparsity and predictive accuracy. LASSO's ability to produce interpretable, sparse models makes it a powerful tool in modern statistical learning, especially when dealing with datasets where the number of predictors may be large relative to the number of observations.

17.9.1 Lasso as a Scale Mixture

The Laplace distribution can be represented as a scale mixture of Normal distributions (Andrews and Mallows 1974):

$$\begin{aligned}\beta_j \mid \sigma^2, \tau_j &\sim N(0, \tau_j^2 \sigma^2) \\ \tau_j^2 \mid \alpha &\sim \text{Exp}(\alpha^2/2) \\ \sigma^2 &\sim \pi(\sigma^2).\end{aligned}$$

We can show equivalence by integrating out τ_j :

$$\begin{aligned}p(\beta_j \mid \sigma^2, \alpha) &= \int_0^\infty \frac{1}{\sqrt{2\pi\tau_j\sigma^2}} \exp\left(-\frac{\beta_j^2}{2\sigma^2\tau_j^2}\right) \frac{\alpha^2}{2} \exp\left(-\frac{\alpha^2\tau_j^2}{2}\right) d\tau_j \\ &= \frac{\alpha}{2\sigma} \exp\left(-\frac{\alpha|\beta_j|}{\sigma}\right).\end{aligned}$$

Thus it is a Laplace distribution with location 0 and scale α/σ . Representation of Laplace prior is a scale Normal mixture allows us to apply an efficient numerical algorithm for computing samples from the posterior distribution. This algorithms is called a Gibbs sample and it iteratively samples from $\theta | a, y$ and $b | \theta, y$ to estimate joint distribution over $(\hat{\theta}, \hat{b})$. Thus, we so not need to apply cross-validation to find optimal value of b , the Bayesian algorithm does it “automatically”.

17.9.2 More Mixture Representations

The power of scale mixture representations extends beyond the Laplace distribution to many common loss functions used in machine learning and statistics. Nicholas G. Polson and Scott (2011) provide a unified framework showing that various loss functions can be represented as variance-mean Gaussian mixtures, enabling efficient posterior sampling via data augmentation.

Consider the regression or classification setting where $z_i = y_i - x_i^T \beta$ for regression, or $z_i = y_i x_i^T \beta$ for binary classification (with $y_i \in \{-1, +1\}$). The key insight is that many loss functions $f(z_i | \beta, \sigma)$ can be expressed through the variance-mean mixture:

$$p(z_i | \beta, \sigma) = \int_0^\infty \phi(z_i | \mu_z + \kappa_z \omega_i, \sigma^2 \omega_i) dP(\omega_i)$$

where $\phi(\cdot | m, v)$ denotes the normal density with mean m and variance v , and $P(\omega_i)$ is an appropriate mixing distribution. The following table summarizes representations for several important loss functions:

Loss Function	$f(z_i \beta, \sigma)$	κ_z	μ_z	Mixing Distribution
Squared-error	z_i^2 / σ^2	0	0	$P(\omega_i) = \omega_i = 1$
Absolute-error	$\ z_i\ / \sigma$	0	0	Exponential
Cheek loss	$\ z_i\ + (2q - 1)z_i$	$1 - 2q$	0	Generalized inverse Gaussian
Support vector machines	$\max(1 - z_i, 0)$	1	1	Generalized inverse Gaussian
Logistic	$\log(1 + e^{z_i})$	1/2	0	Pólya-Gamma

This unified representation has profound computational implications. By introducing latent variables $\{\omega_i\}$ and $\{\lambda_j\}$ through appropriate equations, the

exponential form $\exp\{-L(\beta)\}$ reduces to a Gaussian linear model with heteroscedastic errors. This enables straightforward Gibbs sampling where each conditional distribution has closed form.

The working response z_i equals $y_i - x_i^T \beta$ for Gaussian regression, or $y_i x_i^T \beta$ for binary classification using logistic regression or support-vector machines. Both σ and τ (from the prior) are hyperparameters typically estimated jointly with β , though they may also be specified by the user or chosen by cross-validation. Importantly, in logistic regression σ does not appear in the model as a hyperparameter.

This framework characterizes the general case for many models. Previous studies (Nicholas G. Polson and Scott 2011) have presented similar results for specific models, including support-vector machines and the powered-logit likelihood. The variance-mean mixture approach provides both theoretical insight into the connections between different loss functions and practical algorithms for efficient Bayesian inference.

17.10 Horseshoe

The horseshoe prior represents a significant advancement in Bayesian sparse estimation, addressing fundamental limitations of the Lasso while maintaining computational tractability (Carlos M. Carvalho, Polson, and Scott 2010). The name derives from the shape of its shrinkage profile, which resembles an inverted horseshoe: it applies minimal shrinkage to large signals while aggressively shrinking noise coefficients toward zero. This behavior makes it particularly attractive for high-dimensional problems where strong sparsity is expected but we want to avoid over-shrinking the true signals.

Unlike the Lasso, which applies constant shrinkage across all coefficient magnitudes, the horseshoe exhibits *adaptive shrinkage*. Small coefficients receive heavy regularization approaching zero, while large coefficients are left nearly unregularized. This is precisely the behavior we desire in sparse estimation: confidently shrink noise to zero while preserving signal fidelity. The horseshoe achieves this through a hierarchical Bayesian structure that places a half-Cauchy prior on local scale parameters.

17.10.1 Mathematical Formulation

The horseshoe prior is defined through a hierarchical scale mixture of normals. For the regression model $y = X\beta + \epsilon$ with $\epsilon \sim N(0, \sigma^2 I)$, the horseshoe prior on coefficients β_j is specified as:

$$\begin{aligned}\beta_j \mid \lambda_j, \tau, \sigma^2 &\sim N(0, \lambda_j^2 \tau^2 \sigma^2) \\ \lambda_j &\sim \text{Cauchy}^+(0, 1) \\ \tau &\sim \text{Cauchy}^+(0, 1)\end{aligned}\tag{17.5}$$

where $\text{Cauchy}^+(0, 1)$ denotes the half-Cauchy distribution (the positive half of a standard Cauchy). The parameter λ_j is the *local* shrinkage parameter for coefficient β_j , while τ is the *global* shrinkage parameter that controls overall sparsity. The product $\kappa_j = \lambda_j \tau$ determines the effective scale of β_j .

The half-Cauchy distribution can be represented as a scale mixture itself:

$$\lambda_j \sim \text{Cauchy}^+(0, 1) \iff \lambda_j^2 \mid \nu_j \sim \text{Inv-Gamma}(1/2, 1/\nu_j), \quad \nu_j \sim \text{Inv-Gamma}(1/2, 1)$$

This hierarchical representation enables efficient Gibbs sampling for posterior inference, as each conditional distribution has a closed form.

17.10.2 Shrinkage Properties

The horseshoe prior induces a shrinkage factor $\kappa_j(\beta_j) = E[\kappa_j \mid \beta_j, y]$ that adapts to the data. For large coefficient values $|\beta_j|$, the shrinkage factor approaches 1 (no shrinkage), while for small values it approaches 0 (complete shrinkage). This can be seen through the marginal prior:

$$p(\beta_j \mid \tau, \sigma^2) = \int_0^\infty N(\beta_j \mid 0, \lambda_j^2 \tau^2 \sigma^2) \cdot \text{Cauchy}^+(\lambda_j \mid 0, 1) d\lambda_j$$

The resulting marginal distribution has extremely heavy tails compared to the Laplace prior used in Lasso. Specifically, the horseshoe has infinite moments, while the Laplace has exponential tails. This heavy-tailed behavior means that large signals are barely regularized, avoiding the systematic bias inherent in ℓ_1 penalties.

The posterior mean under the horseshoe prior can be approximated as:

$$E[\beta_j \mid y] \approx (1 - \kappa_j) \cdot 0 + \kappa_j \cdot \hat{\beta}_j^{\text{OLS}}$$

where $\hat{\beta}_j^{\text{OLS}}$ is the ordinary least squares estimate and $\kappa_j \in (0, 1)$ is the data-dependent shrinkage factor. Unlike Lasso, where the shrinkage is constant across coefficients, κ_j adapts to the observed magnitude of $\hat{\beta}_j^{\text{OLS}}$.

17.10.3 Comparison with Other Priors

The horseshoe prior occupies a distinctive position in the landscape of sparse priors. Consider the shrinkage behavior as a function of the observed coefficient:

- *Ridge* (ℓ_2): Shrinkage factor $\kappa_j = 1/(1 + \lambda)$ is constant, providing uniform shrinkage regardless of signal strength
- *Lasso* (ℓ_1): Shrinkage is constant for small coefficients and linear for large ones, $\kappa_j \approx 1 - \lambda/|\beta_j|$ for large $|\beta_j|$
- *Horseshoe*: Shrinkage factor varies from 0 for noise to nearly 1 for signals, $\kappa_j \approx (\lambda_j^2 \tau^2)/(\lambda_j^2 \tau^2 + \sigma^2)$

The global-local structure separates two tasks: τ determines how many coefficients should be non-zero (controlling overall sparsity), while each λ_j determines whether its specific coefficient belongs to the signal or noise group. This separation leads to superior performance in recovering sparse signals compared to the Lasso, particularly when true coefficients vary substantially in magnitude.

17.10.4 Computational Implementation

The horseshoe prior admits efficient posterior sampling through Gibbs sampling, leveraging its hierarchical structure. The full conditional distributions are:

$$\begin{aligned} (\beta | y, \lambda, \tau, \sigma^2) &\sim N((X'X + D_\lambda^{-1})^{-1}X'y, \sigma^2(X'X + D_\lambda^{-1})^{-1}) \\ (\sigma^2 | y, \beta) &\sim \text{Inv-Gamma}\left(\frac{n}{2}, \frac{1}{2}\|y - X\beta\|^2\right) \\ (\lambda_j^2 | \beta_j, \tau, \sigma^2, \nu_j) &\sim \text{Inv-Gamma}\left(1, \frac{1}{\nu_j} + \frac{\beta_j^2}{2\tau^2\sigma^2}\right) \\ (\tau^2 | \beta, \lambda, \sigma^2, \xi) &\sim \text{Inv-Gamma}\left(\frac{p+1}{2}, \frac{1}{\xi} + \frac{1}{2\sigma^2} \sum_{j=1}^p \frac{\beta_j^2}{\lambda_j^2}\right) \end{aligned}$$

where $D_\lambda = \text{diag}(\lambda_1^2 \tau^2, \dots, \lambda_p^2 \tau^2)$ and ν_j, ξ are auxiliary variables from the scale mixture representation of the half-Cauchy.

Example 17.6 (Horseshoe Prior for Sparse Regression). We demonstrate the horseshoe prior's ability to recover sparse signals in a high-dimensional regression setting. Consider a scenario with $n = 100$ observations and $p = 50$ predictors, where only 5 coefficients are truly non-zero with varying magnitudes.

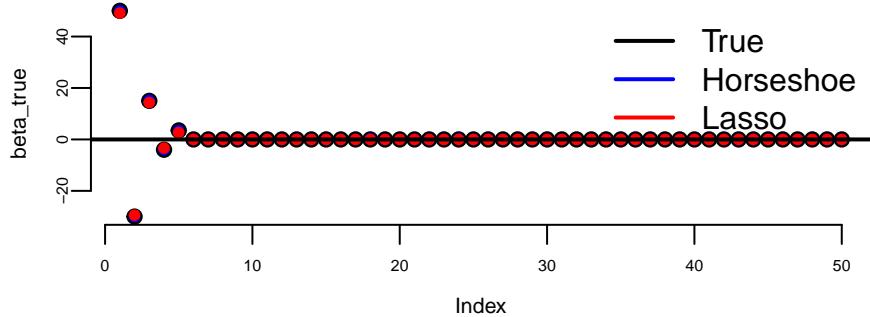


Figure 17.9: Comparison of true, Horseshoe, and Lasso estimates

Figure 17.9 reveals the distinctive shrinkage behaviors of the horseshoe prior compared to the Lasso penalty. The figure displays coefficient estimates (vertical axis) across all 50 predictors (horizontal axis), with black points indicating the true sparse signal where only the first five coefficients are non-zero with magnitudes ranging from 50 to -30. The horseshoe estimates (blue) exhibit a remarkable ability to preserve the large true coefficients nearly intact while aggressively shrinking the truly zero coefficients toward zero, demonstrating minimal bias for strong signals. In contrast, the Lasso estimates (red) systematically undershrink the large coefficients while leaving small spurious non-zero estimates scattered across many truly null predictors. This comparison illustrates the horseshoe prior's theoretical advantage: its aggressive shrinkage of noise parameters combined with minimal penalization of signal parameters, resulting from the heavy tails of the half-Cauchy prior that allow large coefficients to escape shrinkage while the sharp peak near zero effectively eliminates noise.

The quantitative comparison in Table 17.3 confirms the theoretical advantages of the horseshoe prior observed in Figure 17.9. The horseshoe achieves substantially lower coefficient MSE, reflecting its superior recovery of the true sparse signal structure through minimal bias on large coefficients and aggressive shrinkage of noise, while both methods perform reasonably well for prediction MSE since the Lasso's systematic bias in coefficient estimates can be partially compensated by its lower variance in zero coefficients. The horseshoe's adaptive shrinkage through the heavy-tailed half-Cauchy prior provides better overall performance across both metrics, more accurately capturing the underlying signal while effectively suppressing noise. The global shrinkage parameter τ is automatically learned from the data, with posterior mean $\tau^2 \approx 0.005$, reflecting the appropriate level of sparsity and effectively performing automatic model selection without cross-validation.

Table 17.3: Comparison of Coefficient and Prediction Mean Squared Errors

Method	Coefficient.MSE	Prediction.MSE
Horseshoe	0.00	0.52
Lasso	0.05	3.17

The example above shows the horseshoe prior with manual Gibbs sampling implementation for pedagogical purposes. In practice, the `horseshoe` package provides efficient implementations of horseshoe regression that are easier to use. Below we demonstrate a practical application with $n = 50$ observations and $p = 100$ predictors, where only 10 coefficients have true effects.

```
library(horseshoe)
# Create a 50 by 100 design matrix X
X2 <- matrix(rnorm(50 * 100), 50)
# True betas: first 10 are 6 (signals), rest are 0 (noise)
beta2 <- c(rep(6, 10), rep(0, 90))
# Generate response
y2 <- X2 %*% beta2 + rnorm(50)
# Fit Horseshoe model
invisible(capture.output(
  hs.object <- horseshoe(y2, X2, method.tau = "truncatedCauchy",
  ↴ method.sigma = "Jeffreys")
))
```

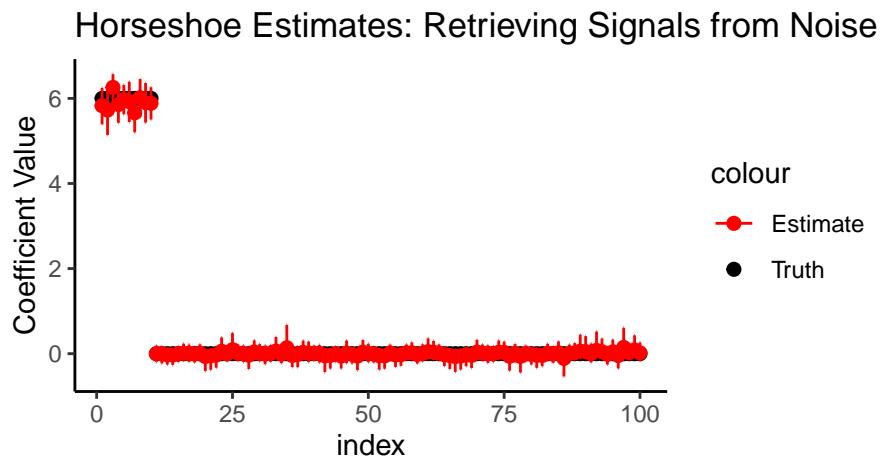
Figure 17.10: Horseshoe prior for sparse regression using `horseshoe` package

Figure 17.10 demonstrates the Horseshoe prior's remarkable ability to distinguish signal from noise in high-dimensional settings. The first 10 coefficients (the true signals with $\beta_j = 6$) are accurately estimated with tight credible intervals around the true values, shown in red. Meanwhile, the remaining 90 noise coefficients (true $\beta_j = 0$) are strongly shrunk toward zero, with their credible intervals tightly concentrated near the origin. This behavior exemplifies the Horseshoe's “selective shrinkage” property: it applies minimal shrinkage to large coefficients (allowing signals to remain unbiased) while aggressively shrinking small coefficients toward zero (effectively removing noise). This makes the Horseshoe particularly well-suited for sparse recovery problems where the number of predictors greatly exceeds the sample size.

We can also perform variable selection by checking if the credible intervals exclude zero.

17.10.5 Comparison and Package Implementation

To illustrate the shrinkage behaviour of the horseshoe, let's plot the posterior mean for β_i as a function of y_i for three different values of τ .

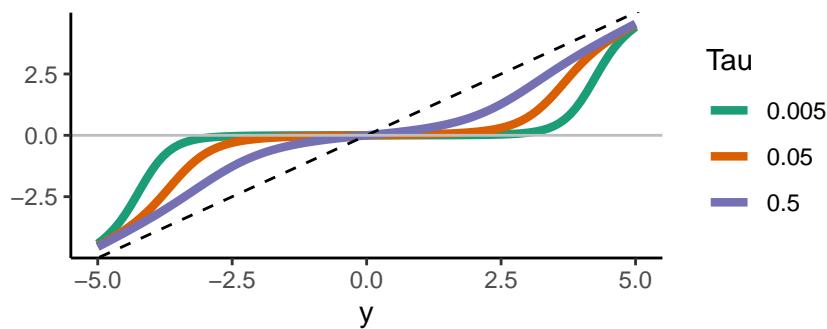


Figure 17.11: Horseshoe posterior mean for three values of tau

Smaller values of τ lead to stronger shrinkage behaviour of the horseshoe. Observations that are in absolute value at most equal to $\sqrt{2\sigma^2 \log(1/\tau)}$ are shrunk to values close to zero (Van der Pas et al (2014)). For larger observed values, the horseshoe posterior mean will tend to the identity (that is, barely any shrinkage, the estimate will be very close to the observed value). The optimal value of τ is the proportion of true signals. This value is typically not known in practice but can be estimated.

The normal means problem

The normal means model is:

$$y_i = \beta_i + \varepsilon_i, \quad i = 1, \dots, n,$$

with ε_i i.i.d. $\mathcal{N}(0, \sigma^2/n_i)$. first, we will be computing the posterior mean only, with known variance σ^2 . The function `HS.post.mean` computes the posterior mean of $(\beta_1, \dots, \beta_n)$.

As an example, we generate 50 data points, the first 10 of which are coming from true signals. The first 10 β_i 's are equal to five and the remaining β_i 's are equal to zero. Then, we estimate τ using the MMLE and use it to find the posterior mean (red). Finally, we use the function `HS.normal.means` to compute posterior means and credible intervals using MCMC.

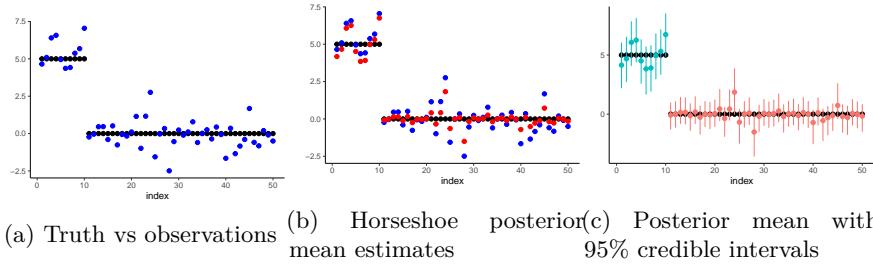


Figure 17.12: Normal means problem. Black = truth, Blue = observations, Red = estimates

The three panels in Figure 17.12 illustrate the horseshoe prior's effectiveness in sparse estimation. Panel (a) shows the challenge: noisy observations (blue) obscure the true sparse signal structure (black), where only the first 10 parameters are non-zero. Panel (b) demonstrates the horseshoe posterior mean estimates (red), which successfully identify the signal locations while shrinking noise toward zero. The horseshoe's adaptive shrinkage becomes most apparent in panel (c), where the full Bayesian analysis provides posterior means with credible intervals. Notice how the horseshoe correctly identifies the signal region (shown in teal) with narrow credible intervals that exclude zero, while the noise region (shown in red) has intervals that include zero. This automatic variable selection through credible intervals exemplifies the horseshoe's oracle-like behavior: it simultaneously performs strong shrinkage on noise components while leaving true signals largely unshrunken, achieving near-optimal estimation without knowing the true sparsity level in advance.

17.11 Bridge (ℓ_α)

The bridge estimator represents a powerful generalization that unifies many popular regularization approaches, bridging the gap between subset selection (ℓ_0) and Lasso (ℓ_1) penalties. For the regression model $y = X\beta + \epsilon$ with unknown vector $\beta = (\beta_1, \dots, \beta_p)'$, the bridge estimator minimizes:

$$Q_y(\beta) = \frac{1}{2} \|y - X\beta\|^2 + \lambda \sum_{j=1}^p |\beta_j|^\alpha \quad (17.6)$$

where $\alpha \in (0, 2]$ is the bridge parameter and $\lambda > 0$ controls the regularization strength. This penalty interpolates between different sparsity-inducing behaviors. As $\alpha \rightarrow 0$, the penalty approaches best subset selection (ℓ_0); when $\alpha = 1$, it reduces to the Lasso penalty (ℓ_1); and when $\alpha = 2$, it becomes the Ridge penalty (ℓ_2). The bridge penalty is non-convex when $0 < \alpha < 1$, making optimization challenging but providing superior theoretical properties. Specifically, when $\alpha < 1$, the penalty is concave over $(0, \infty)$, leading to the oracle property under certain regularity conditions—the ability to identify the true sparse structure and estimate non-zero coefficients as efficiently as if the true model were known.

17.11.1 Bayesian Framework and Data Augmentation

From a Bayesian perspective, the bridge estimator corresponds to the MAP estimate under an exponential-power prior. The Bayesian bridge model treats $p(\beta | y) \propto \exp\{-Q_y(\beta)\}$ as a posterior distribution, arising from assuming a Gaussian likelihood for y and independent exponential-power priors:

$$p(\beta_j | \alpha, \tau) = \frac{\alpha}{2\tau\Gamma(1+1/\alpha)} \exp\left(-\left|\frac{\beta_j}{\tau}\right|^\alpha\right) \quad (17.7)$$

where $\tau = \lambda^{-1/\alpha}$ is the scale parameter. The Bayesian framework offers compelling advantages over classical bridge estimation. Rather than providing only a point estimate, it yields the full posterior distribution, enabling uncertainty quantification and credible intervals. The regularization parameter λ can be learned from the data through hyperpriors, avoiding cross-validation. Most importantly, the bridge posterior is often multimodal, especially with correlated predictors, and MCMC naturally explores all modes while optimization may get trapped in local optima.

Posterior inference for the Bayesian bridge is facilitated by two key data augmentation representations. The first represents the exponential-power prior as a scale mixture of normals using Bernstein's theorem: $\exp(-|t|^\alpha) = \int_0^\infty e^{-st^{2/\alpha}} g(s) ds$, where $g(s)$ is the density of a positive $\alpha/2$ -stable random variable. However, the conditional posterior for the mixing variables becomes an exponentially tilted stable distribution, which lacks a closed form and requires specialized sampling algorithms.

A novel alternative representation avoids stable distributions by expressing the exponential-power prior as a scale mixture of triangular (Bartlett-Fejer) kernels:

$$\begin{aligned} (y | \beta, \sigma^2) &\sim N(X\beta, \sigma^2 I) \\ p(\beta_j | \tau, \omega_j, \alpha) &= \frac{1}{\tau \omega_j^{1/\alpha}} \left\{ 1 - \left| \frac{\beta_j}{\tau \omega_j^{1/\alpha}} \right| \right\}_+ \\ (\omega_j | \alpha) &\sim \frac{1+\alpha}{2} \cdot \text{Gamma}(2+1/\alpha, 1) + \frac{1-\alpha}{2} \cdot \text{Gamma}(1+1/\alpha, 1) \end{aligned} \quad (17.8)$$

where $\{a\}_+ = \max(a, 0)$. This mixture of gamma distributions is much simpler to sample from and naturally captures the bimodality of the bridge posterior through its two-component structure. The choice of representation depends on the design matrix structure: the Bartlett-Fejer representation is 2-3 times more efficient for orthogonal designs, while the scale mixture of normals performs better for collinear designs.

The bridge prior with $\alpha < 1$ satisfies the oracle property under regularity conditions, correctly identifying the true sparsity pattern while avoiding over-shrinkage of large signals through its heavier-than-exponential tails and re-descending score function. The Bartlett-Fejer representation enables an efficient Gibbs sampler that marginalizes over local scale parameters, achieving excellent mixing properties, though a distinctive feature is the posterior's multimodality with correlated predictors, which reflects genuine model uncertainty that the Bayesian approach properly accounts for by averaging over all modes. Extensive simulations demonstrate dramatic improvements over classical methods: with $p = 100$, $n = 101$, and $\alpha = 0.5$, mean squared error was 2254 for least squares, 1611 for classical bridge, and only 99 for Bayesian bridge, with similar gains on benchmark datasets often exceeding 50% reduction in prediction error. Practical guidelines recommend $\alpha \in [0.5, 0.8]$ or learning from data with a uniform prior, using $\text{Gamma}(2, 2)$ for the global scale τ , selecting the Bartlett-Fejer representation for nearly orthogonal designs and scale mixture of normals for collinear predictors, and examining posterior multimodality to understand model uncertainty. The framework extends naturally to other likelihoods and is implemented in the R package **BayesBridge**, occupying a unique position in the regularization landscape by providing less bias than Lasso for large coefficients while maintaining sparsity,

achieving variable selection unlike ridge regression, and offering computational advantages over spike-and-slab priors, making it an excellent choice for modern high-dimensional inference problems.

17.12 Full Bayes for Sparsity Shrinkage

Thus far we have considered penalized optimization approaches that yield point estimates through maximum a posteriori (MAP) estimation. While these methods provide computationally efficient solutions for high-dimensional problems, they do not fully quantify uncertainty about which variables should be included in the model. In this section, we explore full Bayesian approaches to variable selection that compute the complete posterior distribution over both parameters and model structure, allowing us to assess uncertainty about sparsity patterns themselves.

17.12.1 Spike-and-Slab Prior

The gold standard for Bayesian variable selection are spike-and-slab priors, also known as Bernoulli-Gaussian mixtures. These priors provide full model uncertainty quantification by explicitly modeling which variables should be included. Consider a linear regression problem

$$y = \beta_1 x_1 + \dots + \beta_p x_p + e, \quad \text{where } e \sim N(0, \sigma^2), \quad -\infty \leq \beta_i \leq \infty.$$

Our goal is to identify which input variables x_i should be included in the model, seeking a sparse solution where $\|\beta\|_0 = k \ll p$, with $\|\beta\|_0 \stackrel{\text{def}}{=} \#\{i : \beta_i \neq 0\}$ denoting the cardinality of the support of β , also known as the ℓ_0 (pseudo)norm. While continuous shrinkage priors like the Lasso and horseshoe adaptively shrink coefficients, spike-and-slab priors take a more explicit approach by placing positive probability mass at exactly zero.

Under spike-and-slab, each β_i exchangeably follows a mixture prior consisting of δ_0 , a point mass at 0, and a Gaussian distribution centered at zero. Hence we write,

$$\beta_i | \theta, \sigma_\beta^2 \sim (1 - \theta)\delta_0 + \theta N(0, \sigma_\beta^2).$$

Here $\theta \in (0, 1)$ controls the overall sparsity in β and σ_β^2 accommodates non-zero signals. This family is termed as the Bernoulli-Gaussian mixture model in the signal processing community.

A useful re-parameterization, the parameters β is given by two independent random variable vectors $\gamma = (\gamma_1, \dots, \gamma_p)'$ and $\alpha = (\alpha_1, \dots, \alpha_p)'$ such that $\beta_i = \gamma_i \alpha_i$, with probabilistic structure

$$\begin{aligned}\gamma_i | \theta &\sim \text{Bernoulli}(\theta) ; \\ \alpha_i | \sigma_\beta^2 &\sim N(0, \sigma_\beta^2) .\end{aligned}$$

Since γ_i and α_i are independent, the joint prior density becomes

$$p(\gamma_i, \alpha_i | \theta, \sigma_\beta^2) = \theta^{\gamma_i} (1 - \theta)^{1 - \gamma_i} \frac{1}{\sqrt{2\pi\sigma_\beta^2}} \exp\left\{-\frac{\alpha_i^2}{2\sigma_\beta^2}\right\}, \quad \text{for } 1 \leq i \leq p .$$

The indicator $\gamma_i \in \{0, 1\}$ can be viewed as a dummy variable to indicate whether β_i is included in the model.

Let $S = \{i : \gamma_i = 1\} \subseteq \{1, \dots, p\}$ be the “active set” of γ , and $\|\gamma\|_0 = \sum_{i=1}^p \gamma_i$ be its cardinality. The joint prior on the vector $\{\gamma, \alpha\}$ then factorizes as

$$\begin{aligned}p(\gamma, \alpha | \theta, \sigma_\beta^2) &= \prod_{i=1}^p p(\alpha_i, \gamma_i | \theta, \sigma_\beta^2) \\ &= \theta^{\|\gamma\|_0} (1 - \theta)^{p - \|\gamma\|_0} (2\pi\sigma_\beta^2)^{-\frac{p}{2}} \exp\left\{-\frac{1}{2\sigma_\beta^2} \sum_{i=1}^p \alpha_i^2\right\} .\end{aligned}$$

Let $X_\gamma \stackrel{\text{def}}{=} [X_{il}]_{i \in S}$ be the set of “active explanatory variables” and $\alpha_\gamma \stackrel{\text{def}}{=} (\alpha_i)_{i \in S}'$ be their corresponding coefficients. We can write $X\beta = X_\gamma \alpha_\gamma$. The likelihood can be expressed in terms of γ, α as

$$p(y | \gamma, \alpha, \theta, \sigma_e^2) = (2\pi\sigma_e^2)^{-\frac{n}{2}} \exp\left\{-\frac{1}{2\sigma_e^2} \|y - X_\gamma \alpha_\gamma\|_2^2\right\} .$$

Under this re-parameterization by $\{\gamma, \alpha\}$, the posterior is given by

$$\begin{aligned}p(\gamma, \alpha | \theta, \sigma_\beta^2, \sigma_e^2, y) &\propto p(\gamma, \alpha | \theta, \sigma_\beta^2) p(y | \gamma, \alpha, \theta, \sigma_e^2) \\ &\propto \exp\left\{-\frac{1}{2\sigma_e^2} \|y - X_\gamma \alpha_\gamma\|_2^2 - \frac{1}{2\sigma_\beta^2} \|\alpha\|_2^2 - \log\left(\frac{1-\theta}{\theta}\right) \|\gamma\|_0\right\} .\end{aligned}$$

Our goal then is to find the regularized maximum a posterior (MAP) estimator

$$\arg \max_{\gamma, \alpha} p(\gamma, \alpha | \theta, \sigma_\beta^2, \sigma_e^2, y) .$$

By construction, the $\gamma \in \{0, 1\}^p$ will directly perform variable selection. Spike-and-slab priors, on the other hand, will sample the full posterior and calculate the posterior probability of variable inclusion. Finding the MAP estimator is

equivalent to minimizing over $\{\gamma, \alpha\}$ the regularized least squares objective function

$$\min_{\gamma, \alpha} \|y - X_\gamma \alpha\|_2^2 + \frac{\sigma_e^2}{\sigma_\beta^2} \|\alpha\|_2^2 + 2\sigma_e^2 \log\left(\frac{1-\theta}{\theta}\right) \|\gamma\|_0 . \quad (17.9)$$

This objective possesses several interesting properties:

1. The first term is essentially the least squares loss function.
2. The second term looks like a ridge regression penalty and has connection with the signal-to-noise ratio (SNR) $\sigma_\beta^2/\sigma_e^2$. Smaller SNR will be more likely to shrink the estimates towards 0. If $\sigma_\beta^2 \gg \sigma_e^2$, the prior uncertainty on the size of non-zero coefficients is much larger than the noise level, that is, the SNR is sufficiently large, this term can be ignored. This is a common assumption in spike-and-slab framework in that people usually want $\sigma_\beta \rightarrow \infty$ or to be “sufficiently large” in order to avoid imposing harsh shrinkage to non-zero signals.
3. If we further assume that $\theta < \frac{1}{2}$, meaning that the coefficients are known to be sparse *a priori*, then $\log((1-\theta)/\theta) > 0$, and the third term can be seen as an ℓ_0 regularization.

Therefore, our Bayesian objective inference is connected to ℓ_0 -regularized least squares, which we summarize in the following proposition.

(Spike-and-slab MAP & ℓ_0 regularization)

For some $\lambda > 0$, assuming $\theta < \frac{1}{2}$, $\sigma_\beta^2 \gg \sigma_e^2$, the Bayesian MAP estimate defined by Equation 17.9 is equivalent to the ℓ_0 regularized least squares objective, for some $\lambda > 0$,

$$\min_{\beta} \frac{1}{2} \|y - X\beta\|_2^2 + \lambda \|\beta\|_0 . \quad (17.10)$$

First, assuming that

$$\theta < \frac{1}{2}, \quad \sigma_\beta^2 \gg \sigma_e^2, \quad \frac{\sigma_e^2}{\sigma_\beta^2} \|\alpha\|_2^2 \rightarrow 0 ,$$

gives us an objective function of the form

$$\min_{\gamma, \alpha} \frac{1}{2} \|y - X_\gamma \alpha\|_2^2 + \lambda \|\gamma\|_0 , \quad \text{where } \lambda \stackrel{\text{def}}{=} \sigma_e^2 \log((1-\theta)/\theta) > 0 . \quad (17.11)$$

This can be seen as a variable selection version of equation Equation 17.10. To show this, we need only to check that the optimal solution corresponds to a feasible solution and vice versa. This is explained as follows.

On the one hand, assuming $\hat{\beta}$ is an optimal solution, then we can correspondingly define $\hat{\gamma}_i \stackrel{\text{def}}{=} I\{\hat{\beta}_i \neq 0\}$, $\hat{\alpha}_i \stackrel{\text{def}}{=} \hat{\beta}_i$, such that $\{\hat{\gamma}, \hat{\alpha}\}$ is feasible and gives the same objective value as $\hat{\beta}$.

On the other hand, assuming $\{\hat{\gamma}, \hat{\alpha}\}$ is optimal, implies that we must have all of the elements in $\hat{\alpha}_{\gamma}$ should be non-zero, otherwise a new $\tilde{\gamma}_i \stackrel{\text{def}}{=} I\{\hat{\alpha}_i \neq 0\}$ will give a lower objective value. As a result, if we define $\hat{\beta}_i \stackrel{\text{def}}{=} \hat{\gamma}_i \hat{\alpha}_i$, $\hat{\beta}$ will be feasible and gives the same objective value as $\{\hat{\gamma}, \hat{\alpha}\}$.

17.13 Subset Selection (ℓ_0 Norm)

The ℓ_0 norm directly counts the number of non-zero parameters, making it the most natural penalty for variable selection. However, ℓ_0 -regularized optimization problems are NP-hard due to their combinatorial nature. The optimization problem is:

$$\min_{\beta} \frac{1}{2} \|y - X\beta\|_2^2 + \lambda \|\beta\|_0$$

where $\|\beta\|_0 = \#\{j : \beta_j \neq 0\}$ is the number of non-zero coefficients. This directly penalizes model complexity by limiting the number of active predictors.

17.13.1 Connection to Spike-and-Slab Priors

A remarkable connection exists between Bayesian spike-and-slab priors and ℓ_0 regularization. Consider the spike-and-slab prior where each coefficient follows:

$$\beta_j | \theta, \sigma_{\beta}^2 \sim (1 - \theta)\delta_0 + \theta N(0, \sigma_{\beta}^2)$$

Here $\theta \in (0, 1)$ controls the sparsity level and σ_{β}^2 governs the size of non-zero coefficients. This can be reparametrized using indicator variables $\gamma_j \in \{0, 1\}$ and continuous coefficients α_j :

$$\begin{aligned} \beta_j &= \gamma_j \alpha_j \\ \gamma_j | \theta &\sim \text{Bernoulli}(\theta) \\ \alpha_j | \sigma_{\beta}^2 &\sim N(0, \sigma_{\beta}^2) \end{aligned}$$

The maximum a posteriori (MAP) estimator under this prior yields the objective:

$$\min_{\gamma, \alpha} \|y - X_\gamma \alpha_\gamma\|_2^2 + \frac{\sigma^2}{\sigma_\beta^2} \|\alpha\|_2^2 + 2\sigma^2 \log\left(\frac{1-\theta}{\theta}\right) \|\gamma\|_0$$

where X_γ contains only the columns corresponding to $\gamma_j = 1$. Under the assumptions $\theta < 1/2$ (favoring sparsity) and $\sigma_\beta^2 \gg \sigma^2$ (weak shrinkage on non-zero coefficients), this reduces to the ℓ_0 -regularized least squares with $\lambda = 2\sigma^2 \log\left(\frac{1-\theta}{\theta}\right)$.

17.13.2 Single Best Replacement (SBR) Algorithm

Since exact ℓ_0 optimization is intractable, practical algorithms focus on finding good local optima. The Single Best Replacement (SBR) algorithm addresses the fundamental challenge in sparse regression: finding the optimal subset of predictors when the search space is exponentially large. For p predictors, there are 2^p possible subsets to consider, making exhaustive search computationally prohibitive for even moderate p .

Rather than searching over all possible coefficient vectors β , SBR reformulates the ℓ_0 -regularized problem as a discrete optimization over active sets $S \subseteq \{1, 2, \dots, p\}$:

$$\min_S f(S) = \frac{1}{2} \|y - X_S \hat{\beta}_S\|_2^2 + \lambda |S|$$

where $\hat{\beta}_S = (X_S^T X_S)^{-1} X_S^T y$ is the least squares solution on the active set S . This reformulation creates a natural bias-variance tradeoff where larger models with bigger active sets reduce bias but increase the penalty, while smaller models reduce the penalty but may increase bias.

The SBR algorithm operates through a systematic iterative process. The initialization phase begins with an empty active set $S_0 = \emptyset$, computes the initial objective $f(S_0) = \frac{1}{2} \|y\|_2^2$ (corresponding to no predictors), and sets the iteration counter $k = 0$.

The main iteration loop proceeds as follows for each iteration k . During candidate generation, the algorithm considers each variable $j \in \{1, \dots, p\}$ and defines the single replacement operation:

$$S_k \cdot j = \begin{cases} S_k \cup \{j\} & \text{if } j \notin S_k \text{ (addition)} \\ S_k \setminus \{j\} & \text{if } j \in S_k \text{ (removal)} \end{cases}$$

For objective evaluation, each candidate $S_k \cdot j$ is assessed by computing:

$$f(S_k \cdot j) = \frac{1}{2} \|y - X_{S_k \cdot j} \hat{\beta}_{S_k \cdot j}\|_2^2 + \lambda |S_k \cdot j|$$

The best replacement selection identifies:

$$j^* = \arg \min_{j \in \{1, \dots, p\}} f(S_k \cdot j)$$

Finally, the improvement check determines whether to accept the move: if $f(S_k \cdot j^*) < f(S_k)$, the algorithm accepts the move and sets $S_{k+1} = S_k \cdot j^*$; otherwise, it stops and returns S_k as the final solution.

Unlike pure forward selection which only adds variables or backward elimination which only removes variables, SBR can both add and remove variables at each step. This bidirectionality provides substantial advantages. The algorithm can escape local optima by correcting early mistakes through the removal of previously selected variables. When variables are correlated, the algorithm can swap between equivalent predictors to find better solutions. Additionally, the adaptive model size capability allows the algorithm to both grow and shrink the model as needed during the optimization process.

When compared with standard stepwise methods, the advantages become clear. Forward selection uses greedy addition only and can become trapped if early selections are poor. Backward elimination starts with the full model, making it computationally expensive for large p . Traditional forward-backward approaches use separate forward and backward phases, while SBR provides a unified framework that considers both additions and removals at each step.

The key computational challenge lies in evaluating $f(S \cdot j)$ for all p variables at each iteration, as naive implementation would require p separate least squares computations per iteration. Efficient matrix updates provide the solution to this challenge.

For the addition case where $j \notin S$, adding variable j to active set S employs rank-one updates to the Cholesky decomposition. If $X_S^T X_S = L_S L_S^T$, then updating for $X_{S \cup \{j\}}^T X_{S \cup \{j\}}$ requires only $O(|S|^2)$ operations instead of $O(|S|^3)$. Similarly, for the removal case where $j \in S$, removing variable j from active set S uses rank-one downdates to the Cholesky decomposition with similar $O(|S|^2)$ complexity.

The overall computational complexity analysis reveals that each iteration requires $O(p|S|^2)$ operations where $|S|$ is the current active set size, the total number of iterations is typically $O(|S_{final}|)$ in practice, and the overall complexity becomes $O(p|S|^3)$, which is much more efficient than exhaustive search requiring $O(2^p)$ operations.

Theoretical Properties

The convergence properties of SBR are well-established. The algorithm demonstrates finite convergence, provably converging in finite steps since there are only finitely many possible active sets. It exhibits monotonic improvement where the objective function decreases or stays the same at each iteration. Finally, the algorithm achieves local optimality, with the final solution satisfying local optimality conditions.

SBR can be viewed as a coordinate-wise proximal gradient method, establishing a connection to proximal gradient methods. The proximal operator for the ℓ_0 norm is:

$$\text{prox}_{\lambda \|\cdot\|_0}(z) = z \odot \mathbf{1}_{|z| > \sqrt{2\lambda}}$$

This hard thresholding operation is exactly what SBR implements in a coordinate-wise manner.

Under certain regularity conditions, SBR can achieve statistical consistency across multiple dimensions. It demonstrates variable selection consistency by correctly identifying the true active set with high probability. The algorithm provides estimation consistency through consistent estimates of the non-zero coefficients. Additionally, it achieves prediction consistency by attaining optimal prediction error rates.

Practical Implementation Considerations

Several practical considerations affect SBR implementation. For regularization parameter selection, cross-validation provides the standard approach but is computationally expensive, while information criteria such as BIC and AIC can provide faster alternatives. Stability selection offers another approach by running SBR on bootstrap samples and selecting stable variables.

Initialization strategies vary in their effectiveness. The empty start approach using $S_0 = \emptyset$ is most common, while forward start begins with forward selection for a few steps. Random start employs multiple random initializations for better global search capabilities.

Handling numerical issues requires attention to several factors. Multicollinearity concerns necessitate checking condition numbers of $X_S^T X_S$. Rank deficiency situations require handling cases where X_S is rank deficient. Numerical stability can be improved by using QR decomposition instead of normal equations when needed.

Statistical Properties and Performance

Empirical studies demonstrate that SBR achieves statistical performance comparable to the gold-standard spike-and-slab priors while being orders of mag-

nitude faster. The algorithm shows superior variable selection performance compared to Lasso and elastic net in high-correlation settings, achieves lower mean squared error than convex relaxation methods for estimation accuracy, and provides better recovery of the true sparse structure compared to ℓ_1 penalties for sparsity detection.

The connection between spike-and-slab priors and ℓ_0 regularization provides theoretical justification for why SBR performs well: it approximates the MAP estimator of a principled Bayesian model while remaining computationally tractable.

Advantages and Limitations

SBR offers several computational and theoretical advantages. The algorithm provides computational efficiency, running much faster than full Bayesian methods. It maintains a theoretical foundation through its principled connection to spike-and-slab priors. The approach demonstrates flexibility in handling various problem sizes and correlation structures. Finally, it produces sparse, interpretable models that enhance model interpretability.

However, SBR also has certain limitations. The algorithm provides no guarantee of global optimality, potentially stopping at local optima. Its greedy nature may lead to suboptimal early decisions that affect the final solution. Performance sensitivity depends heavily on λ selection, requiring careful parameter tuning. Additionally, the algorithm may struggle with highly correlated predictors in certain scenarios.

Extensions and Variations

Several extensions expand SBR's applicability. Grouped SBR extends to group selection by replacing single variables with groups:

$$S \cdot G = \begin{cases} S \cup G & \text{if } G \cap S = \emptyset \\ S \setminus G & \text{if } G \subseteq S \end{cases}$$

Regularization path computation involves computing solutions for a sequence of λ values while using warm starts from previous solutions. Multiple best replacements consider the k best replacements at each step for more thorough search rather than restricting to single best replacement.

Proximal Perspective

The SBR algorithm can be deeply understood through the lens of proximal operators. The proximal operator for the ℓ_0 norm corresponds to *hard thresholding*.

olding:

$$[\text{prox}_{\gamma\lambda\|\cdot\|_0}(v)]_j = v_j \odot \mathbf{1}_{|v_j| > \sqrt{2\gamma\lambda}}$$

This reveals that SBR essentially performs coordinate-wise hard thresholding, where the decision to include or exclude a variable depends on whether its magnitude exceeds a threshold derived from the regularization parameter. This contrasts with the Lasso's soft thresholding (which shrinks all coefficients) and Ridge's uniform shrinkage. This perspective unifies SBR with continuous optimization approaches, showing it implements the "hard" decision boundary required for exact sparsity.

Spike-and-Slab Examples: Bernoulli-Gaussian and Bernoulli-Laplace

To illustrate the practical implications of different spike-and-slab priors in the proximal framework, we examine two important cases: Bernoulli-Gaussian and Bernoulli-Laplace priors. These examples demonstrate how different prior specifications lead to different shrinkage behaviors and associated penalty functions.

For the normal means problem where $y \mid \beta \sim N(\beta, \sigma^2)$ with Bernoulli-Gaussian prior $\beta \sim (1 - \theta)\delta_0 + \theta N(0, \sigma_\beta^2)$, the marginal distribution of y is:

$$y \mid \theta \sim (1 - \theta)N(0, \sigma^2) + \theta N(0, \sigma^2 + \sigma_\beta^2)$$

The posterior mean takes the form:

$$\hat{\beta}^{BG} = w(y)y$$

where the weight function is:

$$w(y) = \frac{\sigma_\beta^2}{\sigma^2 + \sigma_\beta^2} \left(1 + \frac{(1 - \theta)\phi(y/\sigma)}{\theta\phi(y/\sqrt{\sigma^2 + \sigma_\beta^2})\sqrt{1 + \sigma_\beta^2/\sigma^2}} \right)^{-1}$$

and $\phi(\cdot)$ denotes the standard normal density.

For the Bernoulli-Laplace prior $\beta \sim (1 - \theta)\delta_0 + \theta \text{Laplace}(0, b)$, the expressions become more complex, involving the cumulative distribution function of the standard normal distribution.

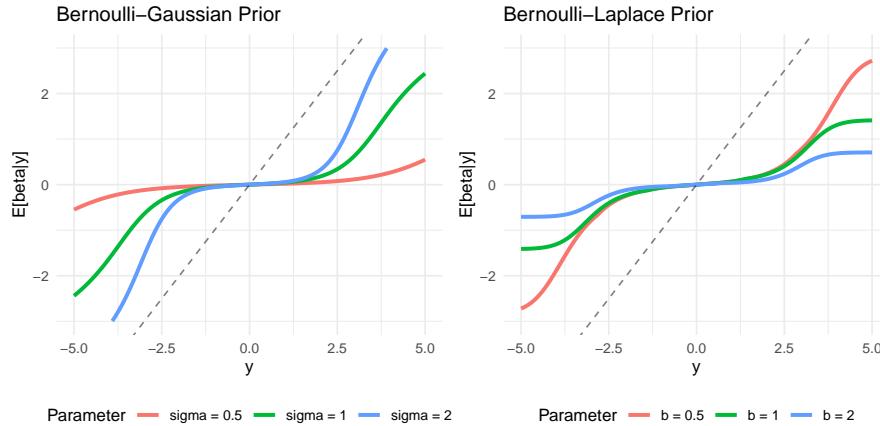


Figure 17.13: Posterior mean functions for Bernoulli-Gaussian (left) and Bernoulli-Laplace (right) priors

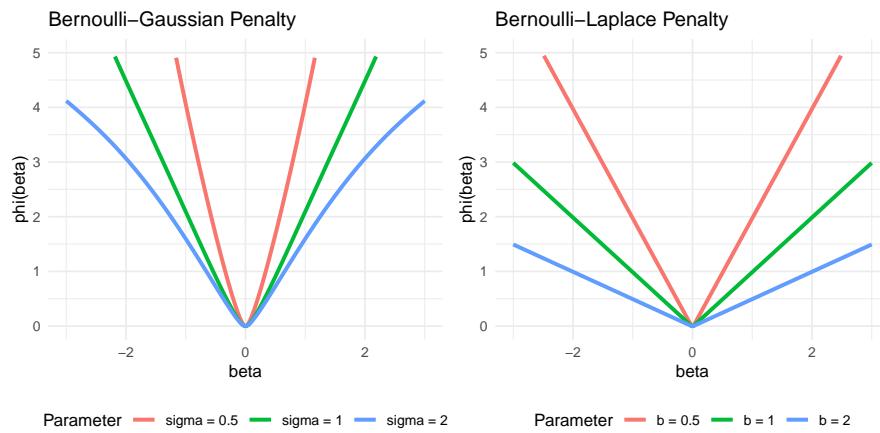


Figure 17.14: Penalty functions associated with Bernoulli-Gaussian (left) and Bernoulli-Laplace (right) posterior means

These figures illustrate several key properties of spike-and-slab priors in the proximal framework. Both priors shrink small observations towards zero, but their behavior differs significantly for larger observations. When the slab variance σ_β^2 (or scale parameter b) is small, Bernoulli-Gaussian priors behave similarly to ridge regression, applying excessive shrinkage to large observations. In contrast, Bernoulli-Laplace priors exhibit behavior more similar to Lasso, with softer shrinkage characteristics.

As the slab parameters increase, both priors approach hard thresholding behavior, and their associated penalty functions ϕ become increasingly similar to non-convex penalties such as SCAD. The penalty functions reveal the “spiky” nature around zero that induces sparsity for small observations, while the different tail behaviors reflect the distinct shrinkage properties of Gaussian versus Laplace slab components.

17.14 Advanced Topics in Regularization

17.14.1 The Vertical Likelihood Duality

For computational efficiency in model evaluation, consider the problem of estimating $\int_{\mathcal{X}} L(x)P(dx)$ where $L : \mathcal{X} \rightarrow \mathbb{R}$. By letting $Y = L(x)$, we can transform this to a one-dimensional integral $\int_0^1 F_Y^{-1}(s)ds$.

We therefore have a duality:

$$\int_{\mathcal{X}} L(x)P(dx) = \int_0^1 F_Y^{-1}(s)ds$$

where Y is a random variable $Y = L(X)$ with $X \sim P(dx)$.

This approach offers several advantages:

- We can use sophisticated grids (Riemann) to approximate one-dimensional integrals
- A grid on inverse CDF space is equivalent to importance weighting in the original \mathcal{X} space
- If $F_Y^{-1}(s)$ is known and bounded, we could use deterministic grids on $[0, 1]$ with $O(N^{-4})$ convergence properties

The main caveats are:

- $F_Y^{-1}(s)$ is typically unknown
- It becomes infinite if L is unbounded
- We often resort to stochastic Riemann sums

The duality implies that finding a good importance function on $[0, 1]$ corresponds to finding good “weighting” in \mathcal{X} space. As a diagnostic for any importance sampling scheme, you should plot the equivalent grid on $[0, 1]$ and estimated values of F_Y^{-1} to assess performance.

17.14.2 Fundamental Integral Identities

The two key integral identities for hyperbolic/GIG (Generalized Inverse Gaussian) and Pólya mixtures are:

$$\frac{\alpha^2 - \kappa^2}{2\alpha} e^{-\alpha|\theta-\mu| + \kappa(\theta-\mu)} = \int_0^\infty \phi(\theta | \mu + \kappa\omega, \omega) p_{\text{gig}}(\omega | 1, 0, \sqrt{\alpha^2 - \kappa^2}) d\omega$$

$$\frac{1}{B(\alpha, \kappa)} \frac{e^{\alpha(\theta-\mu)}}{(1 + e^{\theta-\mu})^{2(\alpha-\kappa)}} = \int_0^\infty \phi(\theta | \mu + \kappa\omega, \omega) p_{\text{pol}}(\omega | \alpha, \alpha - 2\kappa) d\omega$$

where p_{pol} denotes the Pólya density and p_{gig} the GIG density.

17.14.3 Quantile Regression Example

To illustrate these integral identities in practice, consider quantile regression. Start by choosing $q \in (0, 1)$ and define the quantile loss function $l(x) = |x| + (2q-1)x$. This is also known as the check loss or hinge loss function, and is used in quantile regression for the q th quantile Koenker (2005). Johannes, Polson, and Yae (2009) represent this as a pseudo-likelihood involving the asymmetric Laplace distribution.

We can derive the corresponding envelope representation as a variance-mean Gaussian mixture. Let $\kappa = 2q - 1$. Then $g(x) = l(x) + \kappa x = |x|$ is symmetric in x and concave in x^2 . Using the general envelope representation framework, we obtain:

$$l(x) = \inf_{\lambda \geq 0} \left\{ \frac{\lambda}{2} \left(x - \frac{2q-1}{\lambda} \right)^2 - \psi(\lambda) \right\}.$$

where $\psi(\lambda) = \frac{\kappa^2}{2\lambda} - \frac{1}{2\lambda^2}$.

This representation shows how the asymmetric quantile loss can be expressed as an envelope of quadratic functions, connecting it to the Gaussian mixture framework and enabling efficient computational algorithms.

17.14.4 Improper Limit Cases

These expressions lead to three important identities concerning improper limits of GIG and Pólya mixing measures for variance-mean Gaussian mixtures:

$$a^{-1} \exp \{-2c^{-1} \max(a\theta, 0)\} = \int_0^\infty \phi(\theta | -av, cv) dv$$

$$c^{-1} \exp \{-2c^{-1} \rho_q(\theta)\} = \int_0^\infty \phi(\theta | -(2\tau - 1)v, cv) e^{-2\tau(1-\tau)v} dv$$

$$(1 + \exp\{\theta - \mu\})^{-1} = \int_0^\infty \phi(\theta | \mu - \frac{1}{2}v, v) p_{\text{pol}}(v | 0, 1) dv$$

where $\rho_q(\theta) = \frac{1}{2}|\theta| + (q - \frac{1}{2})\theta$ is the check-loss function.

These representations connect to several important regression methods. The first identity relates to *Support Vector Machines* through the hinge loss in the max function. The second identity connects to *Quantile and Lasso Regression* via check-loss and ℓ_1 penalties. The third identity provides the *Logistic Regression* representation.

With GIG and Pólya mixing distributions alone, one can generate the following objective functions:

$$\theta^2, |\theta|, \max(\theta, 0), \frac{1}{2}|\theta| + (\tau - \frac{1}{2})\theta, \frac{1}{1 + e^{-\theta}}, \frac{1}{(1 + e^{-\theta})^r}$$

These correspond to ridge, lasso, support vector machine, check loss/quantile regression, logit, and multinomial models, respectively. More general families can generate other penalty functions—for example, the bridge penalty $|u|^\alpha$ from a stable mixing distribution.

17.14.5 Computational Advantages of Scale Mixtures

The scale mixture representation provides several computational benefits. *Gibbs Sampling* becomes efficient because the conditional distributions in the augmented parameter space are often conjugate, enabling tractable MCMC algorithms. *EM Algorithms* benefit from the E-step involving expectations with respect to the mixing distribution.

This framework has been instrumental in developing efficient algorithms for sparse regression, robust statistics, and non-conjugate Bayesian models. The ability to represent complex penalties as hierarchical Gaussian models bridges the gap between computational tractability and statistical flexibility.

17.14.6 Generalized Ridge Regression in the Canonical Basis

We revisit ridge regression from the perspective of the canonical coordinate system, using the singular value decomposition $X = UDV^T$ and rotated coefficients $\alpha = V^T\beta$. In this basis:

$$Y = X\beta + \epsilon = UDV\alpha + \epsilon$$

Projecting onto U :

$$\hat{\alpha}_i = \alpha_i + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2/d_i^2)$$

With independent priors $\alpha_i \sim N(0, v_i)$, the posterior mean yields a shrinkage estimator:

$$\alpha_i^* = \kappa_i \hat{\alpha}_i, \quad \text{with} \quad \kappa_i = \frac{d_i^2}{d_i^2 + k_i}$$

where $k_i = \sigma^2/v_i$ is the generalized ridge penalty.

This framework offers componentwise optimality and anticipates global-local shrinkage priors where individual parameters receive adaptive regularization.

17.14.7 Trend Filtering and Composite Penalties

Trend filtering provides an example of regularized least squares with composite penalty, useful when a 1D signal has jumps at few positions:

$$\hat{\beta} = \arg \min_{\beta} \left\{ \frac{1}{2} \|y - \beta\|_2^2 + \lambda \phi(D\beta) \right\}$$

where D is the difference matrix and ϕ is a penalty function that encourages sparsity in the differences, leading to piecewise-constant or piecewise-polynomial solutions.

17.15 Final Thoughts

Our exploration began with a sobering revelation: the maximum likelihood estimator, long considered the gold standard of classical statistics, is inadmissible in high-dimensional settings. Stein's paradox demonstrates that for $p \geq 3$ dimensions, there always exist estimators with uniformly lower risk than the MLE. This is not merely a theoretical curiosity—in the normal means problem with $p = 100$, the James-Stein estimator can achieve 67 times lower risk than the MLE. This dramatic improvement illustrates why classical approaches often fail in modern high-dimensional problems and why shrinkage methods have become essential tools in contemporary data science.

The James-Stein estimator's success stems from its ability to “borrow strength” across components through global shrinkage, demonstrating that multivariate estimation problems are fundamentally easier than their univariate counterparts when approached with appropriate regularization. However, global shrinkage alone is insufficient for sparse signals, motivating the development of more sophisticated approaches that can adapt to local signal structure.

A central theme throughout this chapter is the profound duality between Bayesian priors and regularization penalties. Every regularization term $\lambda\phi(f)$

corresponds to a prior distribution through the relationship $\phi(f) = -\log p(f)$, making maximum a posteriori (MAP) estimation equivalent to penalized optimization. This duality provides both theoretical justification for regularization methods and practical guidance for prior specification:

Table 17.4: Penalty types and their corresponding prior distributions.

Method	Penalty	Prior	Key Property/Behavior
Ridge	β^2	Gaussian	Grows Quadratically. Shrinks slowly. Encourages smoothness and numerical stability
Lasso	$ \beta $	Laplace	Grows Linearly. Shrinks strongly. Induces sparsity but shrinks strong signals
Bridge	$ \beta ^\alpha$	Exponential	Interpolates between Ridge and Lasso.
Cauchy	$\log(1 + \beta^2)$	Cauchy	Heavy-tailed. Grows logarithmically. Strong shrinkage near zero, minimal shrinkage for large signals
Horseshoe	$-\log \log(1 + 2\tau^2/\beta^2)$	Horseshoe	Grows almost flat for large signals. Shrinks slowly. Optimal sparsity with strong signals preserved. τ is a scale parameter.
Subset selection	$I(\beta \neq 0)$	Spike-and-slab	Shrinks strongly. Exact sparsity through hard thresholding. Computationally hard.

This mapping reveals why certain penalties work well for particular problem types and guides the selection of appropriate regularization strategies based on problem structure and prior beliefs about the solution. Figure 17.15 shows the unit balls defined by the norms induced by different priors.

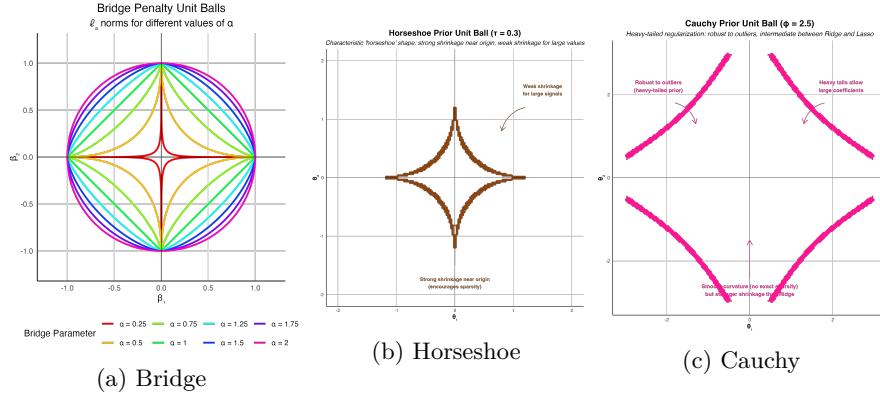


Figure 17.15: Comparison of unit balls for norms induced by different priors

The Figure 17.15a illustrates the unit balls for different penalty norms, each revealing distinct geometric properties. The ℓ_2 (Ridge) penalty creates a circular unit ball that encourages smoothness across all parameters. The ℓ_1 (Lasso) penalty forms a diamond shape that induces sparsity through soft thresholding. Bridge penalties with $\alpha = 0.5$ create more pointed shapes that provide stronger sparsity than Lasso, while $\alpha = 1.5$ produces shapes between the diamond and circle. The ℓ_0 penalty reduces to discrete points on the coordinate axes, representing exact subset selection.

Figure 17.15b shows the unit balls for penalty norms induced by the horseshoe prior. Visually, it looks similar to the unit ball induced by the bridge prior with $\alpha = 0.5$. It demonstrates several remarkable properties that make it particularly effective for sparse estimation problems. Its penalty function $\phi(\theta_i) = -\log(\log(1 + 2\tau^2/\theta_i^2))$ exhibits a unique double-logarithmic structure that creates an ideal balance between sparsity induction and signal preservation.

The horseshoe's most distinctive characteristic is its asymmetric shrinkage behavior. Near the origin, when parameters approach zero, the penalty function grows to infinity, creating extremely strong shrinkage that encourages exact zeros and thus induces sparsity. However, for large parameter values, the penalty grows slowly, allowing large signals to escape penalization with minimal distortion. This behavior is fundamentally different from methods like Lasso, which applies uniform shrinkage regardless of signal magnitude.

The global shrinkage parameter τ provides crucial control over the prior's behavior. Smaller values of τ impose more aggressive shrinkage across all parameters, while larger values become more permissive, allowing signals to emerge more easily. This parameter effectively controls the trade-off between sparsity and signal detection.

From a theoretical perspective, the horseshoe prior achieves optimal minimax

convergence rates of $O(s \log(p/s))$ for s -sparse vectors, making it particularly well-suited for “needle-in-a-haystack” problems where researchers seek to identify a few large signals among many noise variables. Under appropriate regularity conditions, the horseshoe also possesses oracle properties, meaning it can correctly identify the true signal structure.

The geometric visualization reveals why this prior is so effective. The characteristic “horseshoe” shape displays concave contours near the coordinate axes, reflecting strong shrinkage toward zero, while showing convex contours away from the origin, indicating minimal shrinkage for large coefficients. Unlike the Lasso’s diamond shape or Ridge’s circular contours, the horseshoe’s heavy tails allow large coefficients to escape penalization entirely.

This comprehensive visualization demonstrates why the horseshoe prior has become increasingly popular for modern high-dimensional problems where the true signal structure is sparse but individual signals may be large in magnitude. The prior’s ability to provide aggressive shrinkage for noise while preserving signal integrity makes it an ideal choice for contemporary machine learning applications.

Figure 17.15c shows the unit ball for the Cauchy prior. It demonstrates several remarkable properties that make it particularly effective for sparse estimation problems. Its penalty function $\phi(\theta_i) = \log(1+\theta_i^2)$ exhibits a unique logarithmic structure that creates an ideal balance between sparsity induction and signal preservation.

The Cauchy prior’s most distinctive characteristic is its heavy-tailed behavior. Near the origin, when parameters approach zero, the penalty function grows logarithmically, creating extremely strong shrinkage that encourages exact zeros and thus induces sparsity. However, for large parameter values, the penalty grows slowly, allowing large signals to escape penalization with minimal distortion. This behavior is fundamentally different from methods like Lasso, which applies uniform shrinkage regardless of signal magnitude.

The global shrinkage parameter τ provides crucial control over the prior’s behavior. Smaller values of τ impose more aggressive shrinkage across all parameters, while larger values become more permissive, allowing signals to emerge more easily. This parameter effectively controls the trade-off between sparsity and signal detection.

Our analysis reveals a rich spectrum of approaches to sparsity, each with distinct theoretical properties and practical advantages.

Global shrinkage methods such as James-Stein estimation and Ridge regression apply uniform regularization across all parameters. While computationally simple and numerically stable, they cannot adapt to local signal structure and may over-shrink large signals in sparse settings.

Adaptive shrinkage methods like Lasso and horseshoe priors provide differential shrinkage based on signal strength. Lasso employs soft thresholding, while

the horseshoe applies aggressive shrinkage to small signals while preserving large ones, achieving the optimal minimax rate $p_n \log(n/p_n)$. The horseshoe is particularly effective for needle-in-a-haystack problems.

Variable selection methods including spike-and-slab priors and ℓ_0 regularization make discrete inclusion or exclusion decisions. These produce the most interpretable sparse solutions and can achieve oracle properties under appropriate conditions, though they are combinatorially challenging to implement.

Building on the theoretical foundations and geometric intuition developed above, we can distill several actionable guidelines for practitioners and researchers working with regularized models and sparse estimation:

Table 17.5: Practical guidelines for regularized models and sparse estimation.

Scenario	Recommended Methods	Rationale
Low-dimensional ($p < n$)	Ridge regression	Numerical stability; sufficient for moderate p
High-dimensional sparse	Lasso, Bridge ($\alpha \approx 0.7$), Horseshoe	Lasso balances efficiency and sparsity; Bridge and Horseshoe offer better properties
Ultra-sparse	Spike-and-slab, ℓ_0 methods	Optimal performance and exact sparsity
Correlated predictors	Bridge, full Bayesian	Handle correlation better than Lasso
Uncertainty quantification	Full Bayesian methods	Provide credible intervals and posterior probabilities

While our focus has been on linear models, the principles developed here extend broadly to modern AI systems. Deep learning architectures routinely employ regularization techniques (dropout, weight decay, batch normalization) that can be understood through the Bayesian lens developed in this chapter. The success of techniques like variational autoencoders and Bayesian neural networks demonstrates the continued relevance of probabilistic thinking in contemporary machine learning.

Moreover, the sparse estimation techniques discussed here are fundamental to interpretable AI, compressed sensing, and efficient neural architecture design.

The theoretical insights about when different regularization approaches succeed provide guidance for designing and analyzing complex learning systems.

The evolution from maximum likelihood estimation to sophisticated Bayesian regularization represents more than technical progress—it reflects a fundamental shift in how we approach learning from data. Rather than seeking single “best” estimates, modern methods embrace uncertainty, incorporate prior knowledge, and adaptively balance model complexity with empirical fit.

The remarkable fact that Bayesian approaches often dominate classical frequentist methods, even from a frequentist perspective (as demonstrated by Stein’s paradox), suggests that probabilistic thinking provides not just philosophical appeal but concrete practical advantages. In an era of ever-growing data complexity and dimensionality, these theoretical insights become increasingly valuable for developing robust, interpretable, and effective learning algorithms.

The unified framework presented in this chapter—connecting classical statistics, Bayesian inference, optimization theory, and modern machine learning—provides both historical perspective and forward-looking guidance for the continued development of artificial intelligence systems. As we confront increasingly complex learning challenges, from personalized medicine to autonomous systems, the principled approach to regularization and uncertainty quantification developed here will remain fundamental to building trustworthy and effective AI systems.



Part III

Deep Learning



18

Neural Networks

“We have a chance to build machines that will be smarter than us. We need to be thoughtful about what we are asking for.” — Geoffrey Hinton

In 2012, a neural network called AlexNet stunned the computer vision community by winning the ImageNet competition with an error rate nearly half that of the previous year’s winner. This breakthrough marked the beginning of deep learning’s modern era—a period in which neural networks have transformed fields from medical diagnosis to language translation, from game playing to autonomous driving.

The goal of this chapter is to provide a comprehensive overview of deep learning (DL) methods. From a statistical perspective, deep learning extends the framework of generalized linear models by introducing multiple layers of non-linear transformations. While traditional statistical models are well-studied and interpretable, they often lack the flexibility to capture complex input-output relationships. *Black-box* predictive methods, such as decision trees and neural networks, offer greater flexibility at the cost of interpretability. Deep learning excels precisely where high-dimensional problems make traditional model selection challenging.

To understand this transition, consider the architectural trade-off between traditional regression and deep learning. Classical statistical methods follow a *wide and shallow* paradigm: we engineer many features (making the model wide) but apply only a single transformation layer—typically a linear combination followed by a link function, as in GLMs. For instance, polynomial regression with interactions creates a wide feature space ($x_1, x_2, x_1^2, x_2^2, x_1x_2, \dots$) but remains shallow (one layer). In contrast, deep learning adopts a *deep and narrow* architecture: it uses many successive transformations (making the model deep) but potentially fewer units per layer (keeping layers narrow). Rather than manually constructing a vast feature space, deep networks learn hierarchical representations—each layer extracts increasingly abstract features from the layer below. This architectural shift explains why neural networks can model complex relationships without explicit feature engineering: depth provides representational power that compensates for narrower layers, while the hierarchy of transformations discovers features automatically.

Deep learning thus offers a powerful alternative in domains traditionally

served by statistical methods, opening rich avenues for future research—including uncertainty quantification, principled architecture selection, and Bayesian deep learning. Although DL models have been most prominently applied to image analysis and natural language processing, they have also demonstrated superior performance in traditional engineering and scientific domains such as spatio-temporal modeling and financial forecasting (Nicholas Polson, Sokolov, and Xu 2021; Nicholas G. Polson, Sokolov, et al. 2017; M. F. Dixon, Polson, and Sokolov 2019; Sokolov 2017; Bhadra et al. 2021; Behnia, Karbowski, and Sokolov 2023; Nareklishvili, Polson, and Sokolov 2022, 2023b, 2023a; Nicholas G. Polson and Sokolov 2023; Nicholas Polson and Sokolov 2020).

In this chapter, we focus on feed-forward neural networks and their applications to regression and classification. We begin with the mathematical foundations, work through illustrative examples, and discuss practical implementation in **R** and **Python**.

Part II largely relied on feature engineering: transforming raw inputs into predictors that make linear or tree-based models effective. Part III shifts that burden onto the model by learning representations jointly with prediction. This transition is powered by regularization ideas (Chapter 17) and by optimization methods for large parameter spaces (Chapter 20), which make it feasible to fit networks with millions or billions of parameters.

18.1 Introduction

Machine learning and statistical analysis share common foundations, but they emphasize different priorities: ML focuses on scalable algorithms and predictive accuracy, while statistics prioritizes interpretability and inferential rigor. Deep learning sits at the intersection, providing powerful pattern-matching capabilities suitable for AI applications such as image recognition and text analysis.

From a computational perspective, images and text can be represented as high-dimensional matrices and vectors, respectively. The challenge of recognizing objects in images or understanding natural language requires learning complex decision boundaries in these high-dimensional input spaces. Deep learning addresses this challenge by using hierarchical layers of transformations that progressively extract meaningful features from raw data. This approach differs from traditional statistical models in that the features are *learned* from data rather than specified *a priori*.

There are several deep learning architectures - each has its own uses and pur-

poses. Convolutional Neural Networks (CNN) deal with 2-dimensional input objects, i.e., images, and have been shown to outperform other techniques. Recurrent Neural Networks (RNN) have shown the best performance on speech and text analysis tasks.

In general, a neural network can be described as follows. Let f_1, \dots, f_L be given univariate activation functions for each of the L layers. Activation functions are nonlinear transformations of weighted data. A nonlinear activation of an affine transformation is then defined by

$$f_l^{W,b} = f_l \left(\sum_{j=1}^{N_l} W_{lj} X_j + b_l \right) = f_l(W_l X_l + b_l)$$

which implicitly needs the specification of the number of hidden units N_l . Our deep predictor, given the number of layers L , then becomes the composite map

$$\hat{Y}(X) = F(X) = (f_l^{W_1,b_1} \circ \dots \circ f_L^{W_L,b_L})(X).$$

Deep learning's capacity to serve as a universal function approximator has deep mathematical roots—a principle traceable to the functional analysis of the nineteenth century. From a practical perspective, given a large enough dataset, we can empirically learn an optimal predictor. Similar to a classic basis decomposition, the deep approach uses univariate activation functions to decompose a high-dimensional input X .

Let $Z^{(l)}$ denote the l th layer, and so $X = Z^{(0)}$. The final output Y can be numeric or categorical. The explicit structure of a deep prediction rule is then

$$\begin{aligned} \hat{Y}(X) &= W^{(L)} Z^{(L)} + b^{(L)} \\ Z^{(1)} &= f^{(1)}(W^{(0)} X + b^{(0)}) \\ Z^{(2)} &= f^{(2)}(W^{(1)} Z^{(1)} + b^{(1)}) \\ &\dots \\ Z^{(L)} &= f^{(L)}(W^{(L-1)} Z^{(L-1)} + b^{(L-1)}). \end{aligned}$$

Here $W^{(l)}$ is a weight matrix and $b^{(l)}$ are bias terms that shift the activation functions. Designing a good predictor depends crucially on the choice of univariate activation functions $f^{(l)}$. The $Z^{(l)}$ are hidden features (or *latent representations*) that the algorithm extracts from the data.

Put differently, the deep approach employs hierarchical predictors comprising a series of L nonlinear transformations applied to X . Each of the L transformations is referred to as a *layer*, where the original input is X , the output of the first transformation is the first layer, and so on, with the output \hat{Y} as the final layer. The layers 1 to L are called *hidden layers*. The number of layers L represents the depth of our network.

18.2 Mathematical Foundations

Before exploring practical examples, it is instructive to see how neural networks can represent common mathematical operations. Interaction terms such as x_1x_2 and $(x_1x_2)^2$, as well as max functions like $\max(x_1, x_2)$, can be expressed as nonlinear functions of semi-affine combinations. Specifically:

$$x_1x_2 = \frac{1}{4}(x_1 + x_2)^2 - \frac{1}{4}(x_1 - x_2)^2$$

$$\max(x_1, x_2) = \frac{1}{2}|x_1 + x_2| + \frac{1}{2}|x_1 - x_2|$$

$$(x_1x_2)^2 = \frac{1}{4}(x_1 + x_2)^4 + \frac{7}{4 \cdot 3^3}(x_1 - x_2)^4 - \frac{1}{2 \cdot 3^3}(x_1 + 2x_2)^4 - \frac{2^3}{3^3}(x_1 + \frac{1}{2}x_2)^4$$

These identities reveal that products and maxima—operations central to modeling interactions and thresholds—can be constructed from combinations of univariate functions applied to linear projections of the input. This is precisely what neural networks do.

Persi Diaconis and Shahshahani (1981) provides further discussion in the context of *Projection Pursuit Regression*, where the model uses a layered structure $\sum_{i=1}^N f(w_i^\top X)$. Their work on composite iterated functions foreshadowed the modern use of multiple layers to model complex multivariate systems.

18.2.1 ReLU Networks as Max-Sum Networks

Deep ReLU architectures can be viewed as max-sum networks via a simple identity. Define $x^+ = \max(x, 0)$. Let $f_x(b) = (x + b)^+$ where b is an offset. Then $(x + y^+)^+ = \max(0, x, x + y)$. This is generalized in Feller (1971) (p.272), who shows by induction that

$$(f_{x_1} \circ \dots \circ f_{x_k})(0) = (x_1 + (x_2 + \dots + (x_{k-1} + x_k^+)^+)^+)^+ = \max_{1 \leq j \leq k} (x_1 + \dots + x_j)^+$$

A composition of max-layers thus reduces to a single-layer max-sum network. This mathematical insight helps explain why deep ReLU networks are so effective: they can efficiently represent a rich class of piecewise linear functions.

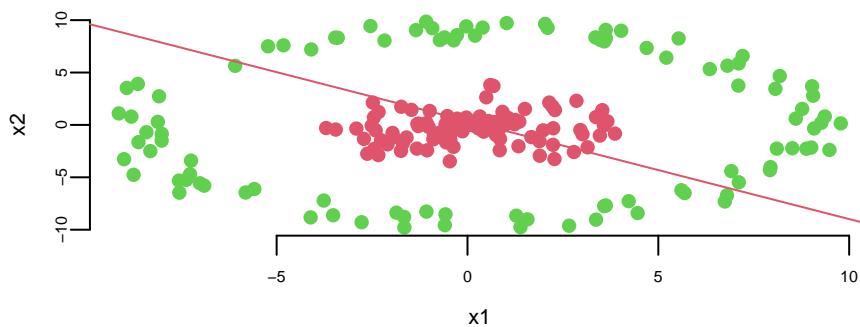
18.3 Classification with Neural Networks

Having established the mathematical framework of layers and activation functions, let us now ground these concepts in a concrete classification problem. We will start with a simple 2D visual example to build intuition before moving to more complex architectures.

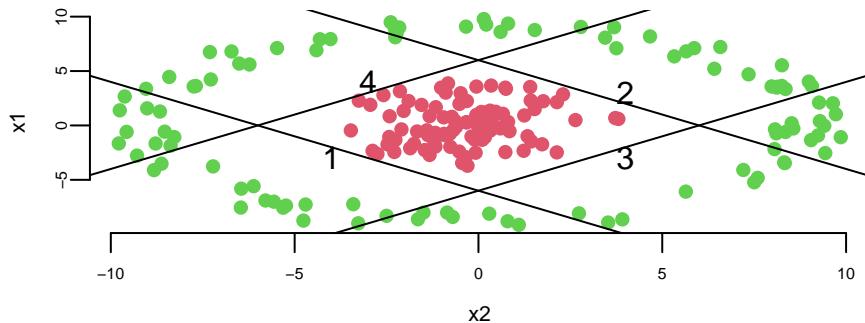
To illustrate the concepts introduced above, we apply a feed-forward neural network with one hidden layer to a binary classification problem. Consider the simulated dataset shown below, where points are generated from two distributions: the inner cluster (green) represents one class, while the outer ring (red) represents the other. The goal is to learn a decision boundary that separates these two classes.

As a baseline, we first attempt to classify the data using logistic regression:

```
# Fit a logistic regression model
fit <- glm(label ~ x1 + x2, data = as.data.frame(d), family =
  binomial(link = "logit"))
# Plot the training dataset
plot(d[, 2], d[, 3], col = d[, 1] + 2, pch = 16, xlab = "x1",
  ylab = "x2")
th <- fit$coefficients
# Plot the decision boundary
abline(-th[1] / th[3], -th[2] / th[3], col = 2)
```



Logistic regression fails to distinguish the classes because the data is not linearly separable. However, we can use multiple lines to separate the data.



Now, we do the same thing as in simple logistic regression and apply logistic function to each of those lines

Using the matrix notation, we have

$$z = \sigma(Wx + b), \quad W = \begin{bmatrix} 1 & 1 \\ -1 & -1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ 6 \\ 6 \\ 6 \end{bmatrix}, \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

The model shown above is the first layer of our neural network. It takes a two-dimensional input x and produces a four-dimensional output z which is called a feature vector. The feature vector is then passed to the output layer, which applies simple logistic regression to the feature vector.

$$\hat{y} = \sigma(w^T z + b), \quad w = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad b = -3.1, \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

The output of the output layer is the probability of the positive class.

We can use our model to do the predictions now

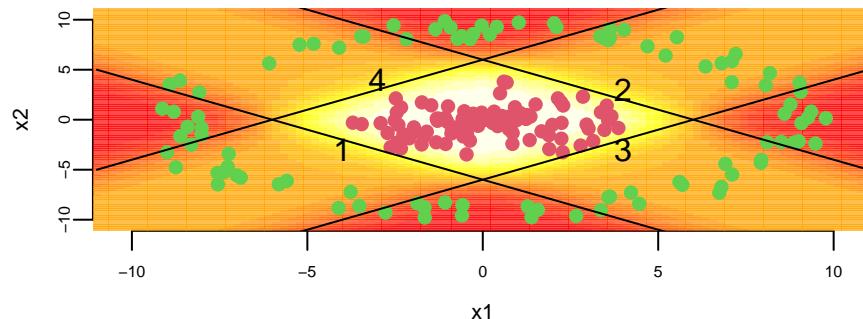
```
## 0.71
## 0.26
```

The model generates sensible predictions, let's plot the decision boundary to see how well it separates the data.

```

x1 <- seq(-11, 11, length.out = 100)
x2 <- seq(-11, 11, length.out = 100)
gr <- as.matrix(expand.grid(x1, x2))
## 10000      2
yhat <- apply(gr, 1, predict_prob)
## 10000
image(x1, x2, matrix(yhat, ncol = 100), col = heat.colors(20,
  0.7))

```



How about a regression model? We will use a one-layer neural network to fit a quadratic function. We simulate noisy data from the following model

$$y = 0.5 + 0.3x^2 + \epsilon, \epsilon \sim N(0, 0.02^2)$$

And use 3 hidden units in the first hidden layer and two units in the second hidden layer. The output layer is a single unit. We will use the hyperbolic tangent (\tanh) activation function for all layers. The model is defined as follows

The hidden layer has three outputs (neurons) and uses the ReLU activation function. The output linear layer has a single output. Thus, the prediction \hat{y} is generated as a linear model of the feature vector z_0 . The model has 8 parameters. Let's generate training data and fit the model. We will use the BFGS optimization algorithm to minimize the loss function (negative log-likelihood) of the model.

```
## -0.24  1.39 -0.84  0.46  0.50  0.18  0.45  0.37
```

Figure 18.1 shows the quadratic function and the neural network model. The solid black line is the neural network model, and the dashed lines are the basis functions. The model fits the data well.

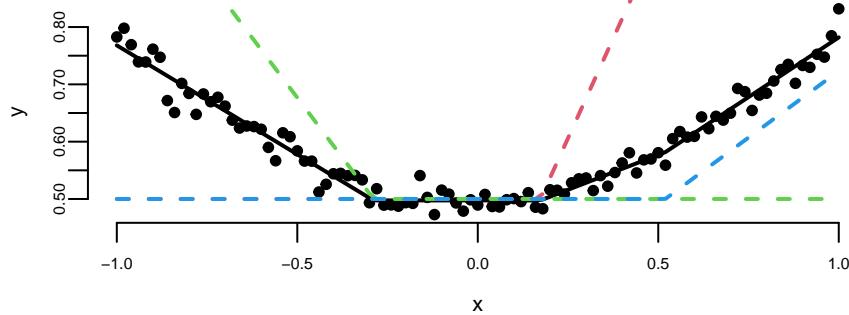


Figure 18.1: Noisy quadratic function approximated by a neural network with ReLU activation function.

Let's try the tanh function

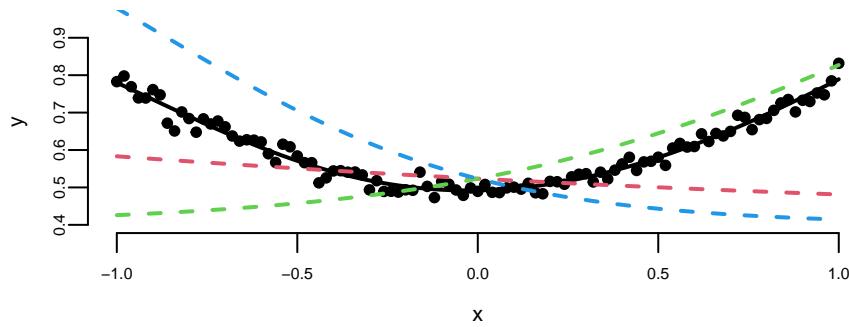


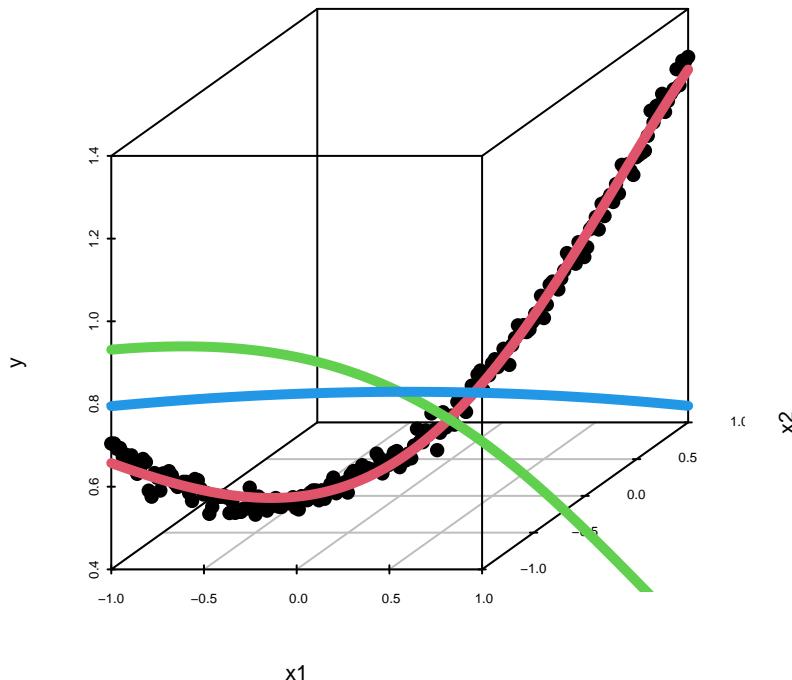
Figure 18.2: Noisy quadratic function approximated by a neural network with tanh activation function.

Notice that we did not need to explicitly specify a quadratic term; the neural network discovered this structure from the data. The model automatically learned an appropriate basis function representation.

The same approach extends naturally to interaction terms. Consider the following data-generating process:

$$y = 0.5 + 0.1x_1 + 0.2x_2 + 0.5x_1x_2 + \epsilon, \quad \epsilon \sim N(0, 0.02^2)$$

We can use the same model as above, but with two input variables. The model will learn the interaction term from the data.



Effectively, neural networks serve as flexible function approximators, analogous to nonparametric regression methods but with the crucial advantage of learning the basis functions directly from data.

18.4 Activation Functions

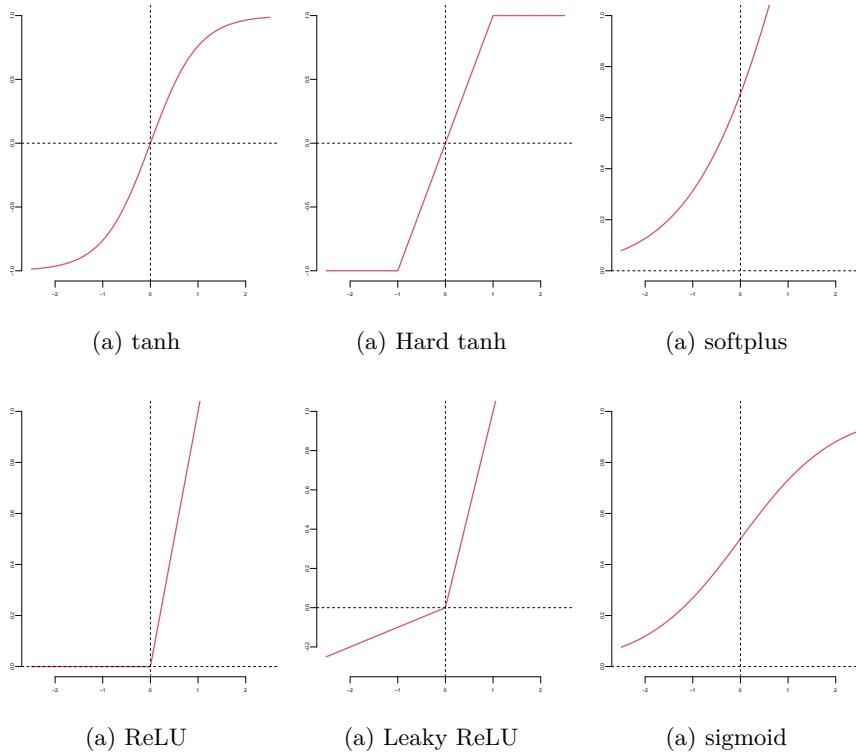
The last output layer of a neural network has sigmoid activation function for binary output variable (classification) and no activation function for continuous output variable regression. The hidden layers can have different activation functions. The most common activation functions are the hyperbolic tangent

function and the rectified linear unit (ReLU) function.

A typical approach is to use the same activation function for all hidden layers. The hyperbolic tangent function is defined as

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Notice that the hyperbolic tangent function is a scaled version of the sigmoid function, with $\tanh(0) = 0$. It is a smooth function which is differentiable everywhere.



A significant limitation of sigmoid and tanh functions is their tendency to saturate at extreme input values, causing gradients to vanish. When we try to learn the weights of the network, the optimization algorithms make small steps in the space of the parameters and when the weights are large the small changes won't affect the values of the layers' outputs and optimization will "stagnate."

Consequently, the gradients become negligible, causing the optimization process to stagnate (the "vanishing gradient" problem). The ReLU function is

defined as

$$\text{ReLU}(z) = \max(0, z)$$

The ReLU function is a piecewise linear function which is computationally efficient and easy to optimize. The ReLU function is the most commonly used activation function in deep learning. The ReLU function is not differentiable at $z = 0$, but it is differentiable everywhere else. The derivative of the ReLU function is

$$\text{ReLU}'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \end{cases}$$

18.5 Unsupervised Learning: Auto-Encoders

Auto-encoding is an important dimensionality reduction technique. An auto-encoder is a neural network architecture designed to replicate its input X , i.e., $Y = X$, via a *bottleneck* structure. The model $F^{W,b}(X)$ learns to compress the information required to recreate X into a lower-dimensional representation. Heaton, Polson, and Witte (2016) provide an application to smart indexing in finance.

Suppose that we have N input vectors $X = \{x_1, \dots, x_N\} \in \mathbb{R}^{M \times N}$ where N is the sample size and M is the feature dimension, and we wish to reconstruct them. Setting biases to zero, for the purpose of illustration, and using only one hidden layer ($L = 2$) with $K < N$ factors, gives for $j = 1, \dots, N$:

$$Y_j(x) = F_W^m(X)_j = \sum_{k=1}^K W_2^{jk} f \left(\sum_{i=1}^N W_1^{ki} x_i \right) = \sum_{k=1}^K W_2^{jk} Z_j \quad \text{for } Z_j = f \left(\sum_{i=1}^N W_1^{ki} x_i \right)$$

Since, in an auto-encoder, we fit the model $X = F_W(X)$, and *train* the weights $W = (W_1, W_2)$ with regularization penalty in a

$$\mathcal{L}(W) = \operatorname{argmin}_W \|X - F_W(X)\|^2 + \lambda \phi(W)$$

with

$$\phi(W) = \sum_{i,j,k} |W_1^{ik}|^2 + |W_2^{jk}|^2.$$

Writing the deep learning objective as an augmented Lagrangian using the Alternating Direction Method of Multipliers (ADMM) with a hidden factor

Z , leads to a two step algorithm: an encoding step (a penalty for Z), and a decoding step for reconstructing the output signal via

$$\operatorname{argmin}_{W,Z} \|X - W_2 Z\|^2 + \lambda \phi(Z) + \|Z - f(W_1, X)\|^2,$$

where the regularization on W_1 induces a penalty on Z . The last term is the encoder, the first two the decoder.

If W_2 is estimated from the structure of the training data matrix, then we have a traditional factor model, and the W_1 matrix provides the factor loadings. Principal Component Analysis (PCA), Partial Least Squares (PLS), and Sliced Inverse Regression (SIR) fall into this category (see Cook 2007 for further discussion). If W_2 is trained based on the pair $\hat{X} = \{Y, X\}$ then we have a sliced inverse regression model. If W_1 and W_2 are simultaneously estimated based on the training data X , then we have a two layer deep learning model.

Auto-encoding demonstrates that deep learning does not directly model variance-covariance matrix explicitly as the architecture is already in predictive form. Given a hierarchical non-linear combination of deep learners, an implicit variance-covariance matrix exists, but that is not the driver of the algorithm.

18.5.1 Dynamic Auto-Encoders

Another interesting area for future research involves applying these concepts to time-series, such as using Long Short Term Memory models (LSTMs). For example, a dynamic one layer auto-encoder for a financial time series (Y_t) is a coupled system of the form

$$Y_t = W_x X_t + W_y Y_{t-1} \quad \text{and} \quad \begin{pmatrix} X_t \\ Y_{t-1} \end{pmatrix} = W Y_t$$

Here, the state equation encodes and the matrix W decodes the Y_t vector into its history Y_{t-1} and the current state X_t .

18.6 Modern Deep Learning Architectures

The evolution of deep learning has led to increasingly sophisticated architectural patterns that go beyond simple stacked layers. Modern systems often combine multiple neural networks, reuse pre-trained components, and employ

ensemble methods to achieve state-of-the-art performance. In this section, we explore three key architectural paradigms that have become central to contemporary deep learning practice: transfer learning with pre-trained blocks, mixture of experts models, and ensemble methods.

18.6.1 Transfer Learning and Pre-trained Blocks

One of the most significant developments in deep learning is the recognition that neural networks learn hierarchical representations, where early layers capture general features and deeper layers encode task-specific patterns. This observation has given rise to *transfer learning*, where models trained on large-scale datasets can be adapted to new tasks with limited data.

Consider the analogy of learning to drive different vehicles. Once you learn to drive a car, adapting to a truck requires learning the differences rather than starting from scratch. Similarly, a neural network trained to recognize objects in millions of images has learned fundamental visual features—edges, textures, shapes—that are useful for many computer vision tasks.

The typical transfer learning workflow involves taking a pre-trained model, removing its final layers, and replacing them with new layers appropriate for the target task. For example, a convolutional neural network (CNN) trained on ImageNet we discuss CNNs in detail in 22 can be adapted for medical image classification by freezing the early convolutional layers and retraining only the final classification layers. The mathematical formulation is straightforward. Let $F_{\text{pre}}(X; \theta_{\text{frozen}})$ represent the pre-trained feature extractor with frozen parameters, and $G(Z; \theta_{\text{new}})$ the new task-specific layers. The complete model becomes:

$$\hat{Y} = G(F_{\text{pre}}(X; \theta_{\text{frozen}}); \theta_{\text{new}})$$

During training, we only optimize θ_{new} while keeping θ_{frozen} fixed. This approach has several advantages. First, it dramatically reduces the amount of labeled data required for the new task. Second, it decreases training time since we only update a subset of parameters. Third, it often leads to better generalization, especially when target task data is limited.

Modern frameworks make transfer learning remarkably accessible. In natural language processing, transformer-based models like BERT (Devlin et al. 2019) and GPT (Radford et al. 2018; Brown et al. 2020) are routinely fine-tuned for downstream tasks such as sentiment analysis, named entity recognition, and question answering. These models, pre-trained on billions of tokens, capture rich linguistic representations that transfer across diverse language tasks [we explore transformers and language models in Chapters 24 and 25].

The practice of using pre-trained blocks extends beyond simple transfer learning. Modern architectures often incorporate multiple pre-trained components as building blocks. For instance, a system for visual question answering might combine a pre-trained CNN for image encoding with a pre-trained transformer for language understanding, with only a small fusion network trained from scratch.

18.6.2 Mixture of Experts

As models grow in scale and complexity, a natural question arises: must every parameter be active for every input? The *mixture of experts* (MoE) architecture provides an elegant answer by using different sub-networks (experts) for different inputs, with a gating mechanism that decides which experts to activate.

The core idea traces back to work by Jacobs et al. (1991), but has seen renewed interest in modern large-scale systems. Think of a medical diagnosis system where different specialists (cardiologist, neurologist, oncologist) examine patients based on their symptoms. A triage system routes each patient to the appropriate specialist. Similarly, in an MoE model, a gating network routes inputs to specialized expert networks.

Formally, let f_1, \dots, f_K denote K expert models (networks), each implementing a function $f_k(X; \theta_k)$. A gating network $g(X; \phi)$ produces a probability distribution over experts, for example:

$$g(X; \phi) = \text{softmax}(W_g X + b_g) \in \mathbb{R}^K$$

The final output is a weighted combination of expert predictions:

$$\hat{Y}(X) = \sum_{k=1}^K g_k(X; \phi) \cdot f_k(X; \theta_k).$$

In practice, to improve computational efficiency, only the top- N experts with highest gating scores are activated for each input, setting others to zero. This sparse activation pattern allows MoE models to have enormous capacity while maintaining reasonable computational costs.

From a statistical perspective, W. Jiang and Tanner (1999a) provided important theoretical foundations for mixture of experts models. They demonstrated that hierarchical mixtures of experts (HME) can approximate arbitrary mean functions at a rate of $O(m^{-2/s})$ in L_p norm, where m is the number of experts and s is the input dimensionality. This result establishes that MoE models are universal approximators with quantifiable convergence rates, connecting them to the broader statistical literature on nonparametric regression.

The work by Stroud, Müller, and Polson (2003) demonstrates an elegant application of mixture-of-experts principles within Bayesian inference for complex time series models. Their approach addresses the fundamental challenge of estimating hidden states in nonlinear state-space models where exact inference is mathematically intractable. The core innovation lies in using an auxiliary mixture model to approximate the intractable nonlinear components. This auxiliary model functions as a mixture of experts, where each expert is a simple linear regression model of the form $p^a(y_t|x_t, z_t = k) = \mathcal{N}(\alpha_k + \beta_k x_t, \tau_k^2)$. These linear experts provide local approximations of the true nonlinear function around specific “knots” in the state space. The gating mechanism employs state-dependent weights $\pi_k(x_t)$ determined by Gaussian kernels centered at different locations. The current hidden state value x_t serves as input to this gating network, which determines the relevance of each linear expert for that particular region of the state space.

A critical theoretical challenge with MoE models is *identifiability*: can the model parameters be uniquely determined from the data? W. Jiang and Tanner (1999b) showed that without constraints, MoE models suffer from inherent non-identifiability due to invariant transformations. For example, permuting expert labels or translating gating parameters can yield identical predictions. They established that by imposing order restrictions on expert parameters (such as requiring experts to be ordered by their intercepts) and proper initialization of gating parameters, MoE systems become identifiable. This work provides the statistical rigor necessary for reliable inference and interpretation of mixture of experts models, complementing the computational advances in modern implementations.

Recent work has demonstrated the power of this approach at scale. Fedus, Zoph, and Shazeer (2022) trained a mixture of experts language model with 1.6 trillion parameters, where only a small fraction are active for any given input. The model achieves strong performance while requiring less computation per token than dense models of comparable quality. Similarly, Riquelme et al. (2021) showed that vision transformers with mixture of experts layers can achieve better accuracy-computation trade-offs than their dense counterparts.

The gating mechanism introduces an interesting learning problem: the network must simultaneously learn to specialize experts and to route inputs appropriately. Various training strategies have been proposed, including auxiliary losses that encourage load balancing across experts and techniques to promote expert specialization (Shazeer et al. 2017).

18.6.3 Ensembles

While mixture of experts learns a single integrated model with internal routing, *ensemble methods* combine predictions from multiple independently trained models. The rationale is straightforward: different models trained on the same

data often make different errors, and averaging their predictions can reduce variance and improve overall performance.

Ensemble methods have a long history in statistics and machine learning. The classical result from Dietterich (2000) shows that an ensemble is effective when individual models are accurate and diverse. Consider M models f_1, \dots, f_M . For regression, a simple ensemble averages predictions:

$$\hat{Y}_{\text{ensemble}}(X) = \frac{1}{M} \sum_{m=1}^M f_m(X)$$

For classification, we can use majority voting or average the predicted probabilities. The bias-variance decomposition provides insight into why ensembles work. The expected squared error of a single model can be written as:

$$\mathbb{E} ((Y - f(X))^2) = \text{Bias}^2 + \text{Variance}$$

If we average M models with the same bias but independent errors, the variance term reduces by a factor of M , assuming the models are uncorrelated. In practice, models are not perfectly independent, but ensembles still typically achieve substantial variance reduction.

In deep learning, several ensemble strategies are commonly employed:

1. *Model averaging*: Train multiple neural networks with different random initializations or architectures, then average their predictions at test time.
2. *Snapshot ensembles* (Huang et al. 2017): Save model checkpoints at various points during training (particularly at local minima) and ensemble them.
3. *Multi-architecture ensembles*: Combine models with different architectures (e.g., CNNs, transformers, and recurrent networks) to capture complementary patterns.

A particularly effective technique is *dropout as ensemble* (Srivastava et al. 2014). During training, dropout randomly deactivates neurons with probability p . At test time, all neurons are active but their outputs are scaled by $(1-p)$. Gal and Ghahramani (2016) showed that this can be interpreted as approximate Bayesian inference, where dropout at test time (called Monte Carlo dropout) produces an ensemble of sub-networks, providing both predictions and uncertainty estimates.

The computational cost of ensembles is their main drawback—infERENCE requires running multiple models. However, the reliability gains are often worth

the expense, particularly in high-stakes applications like medical diagnosis, autonomous driving, and financial forecasting. Modern competitions and benchmarks frequently use ensembles: winning solutions in Kaggle competitions typically combine dozens of models (T. Chen and Guestrin 2016).

18.6.4 Architectural Innovations

These three paradigms—pre-trained blocks, mixture of experts, and ensembles—represent different philosophies for building capable systems. Transfer learning emphasizes knowledge reuse, recognizing that representations learned on large datasets are broadly useful. Mixture of experts emphasizes conditional computation, activating only relevant capacity for each input. Ensembles emphasize diversity and robustness, combining multiple perspectives to improve reliability.

Modern systems often combine these approaches. For instance, a production system might ensemble multiple models, each of which uses pre-trained components and employs mixture of experts layers for efficiency. The field continues to develop new architectural patterns, but these fundamental ideas—reusing learned knowledge, routing computation dynamically, and aggregating diverse predictions—are likely to remain central to deep learning practice.

18.7 Summary

In this chapter, we introduced the mathematical foundations of neural networks and deep learning. We began by establishing that neural networks are hierarchical compositions of nonlinear transformations—a framework that extends generalized linear models while providing the flexibility to learn complex patterns from data. Key concepts covered include:

- *Feedforward architecture*: A neural network applies successive layers of weighted linear combinations followed by nonlinear activation functions, extracting increasingly abstract features at each layer.
- *Activation functions*: Functions like ReLU, tanh, and sigmoid introduce nonlinearity, enabling networks to approximate arbitrary functions. ReLU has become the default choice for hidden layers due to its computational efficiency and favorable gradient properties.
- *Auto-encoders*: These networks learn compressed representations by training to reconstruct their inputs through a bottleneck layer, providing a powerful approach to unsupervised dimensionality reduction.

- *Transfer learning and pre-trained blocks:* Knowledge learned on large datasets can be transferred to new tasks, dramatically reducing data and computation requirements.
- *Mixture of experts:* Conditional computation allows models to activate only relevant sub-networks for each input, enabling massive scale with manageable computational costs.
- *Ensemble methods:* Combining diverse models improves robustness and reduces variance, with techniques ranging from simple averaging to dropout-based approximations.

The implementations presented in this chapter—from simple classification problems to regression with learned basis functions—demonstrate that neural networks can be understood through the lens of traditional statistical methods while offering capabilities that extend far beyond them.

Looking ahead, Chapter 22 covers *convolutional neural networks* (CNNs), which exploit spatial structure in images through weight sharing and local connectivity. Chapter 23 and Chapter 24 introduce *transformers* and *large language models*, architectures that have transformed natural language processing through attention mechanisms and massive scale. These architectures build directly on the principles established in this chapter: layered representations, learned features, and the power of deep hierarchical computation.

19

Theory of Deep Learning

This chapter explores the theoretical foundations of deep learning through the lens of multivariate function approximation. We begin with *ridge functions* as fundamental building blocks—functions of the form $f(x) = g(w^T x)$ that represent one of the simplest forms of nonlinear multivariate functions by combining a single linear projection with a univariate nonlinear transformation. Their key geometric property—remaining constant along directions orthogonal to the projection vector w —makes them particularly useful for high-dimensional approximation.

Building on ridge functions, we introduce *projection pursuit regression*, which approximates complex input-output relationships using linear combinations of ridge functions. This technique, developed in the 1980s, provides the mathematical foundation for understanding how neural networks decompose high-dimensional problems.

The chapter culminates with the *Kolmogorov Superposition Theorem* (KST), a profound result showing that any real-valued continuous function can be represented as a sum of compositions of single-variable functions. This theorem provides a theoretical framework for understanding how multivariate functions can be decomposed into simpler, more manageable components—a principle that underlies the architecture of modern neural networks.

Throughout, we examine a central question: can we achieve superior performance through mathematically elegant representations of multivariate functions rather than raw computational power? This tension between theoretical efficiency and practical computation motivates much of the research in deep learning theory.

19.1 Ridge and Projection Pursuit Regression

To understand the significance of this trade-off, we consider ridge functions, which represent a fundamental building block in multivariate analysis. Since our ultimate goal is to model arbitrary multivariate functions f , we need a

way to reduce dimensionality while preserving the ability to capture nonlinear relationships. Ridge functions accomplish this by representing one of the simplest forms of nonlinear multivariate functions, requiring only a single linear projection and a univariate nonlinear transformation. Formally, a ridge function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ takes the form $f(x) = g(w^T x)$, where g is a univariate function and $x, w \in \mathbb{R}^n$. The non-zero vector w is called the direction. The term “ridge” reflects a key geometric property: the function remains constant along any direction orthogonal to w . Specifically, for any direction u such that $w^T u = 0$, we have

$$f(x + u) = g(w^T(x + u)) = g(w^T x) = f(x)$$

This structural simplicity makes ridge functions particularly useful as building blocks for high-dimensional approximation.

Ridge functions play a central role in high-dimensional statistical analysis. For example, projection pursuit regression approximates input-output relations using a linear combination of ridge functions (Jerome H. Friedman and Stuetzle 1981; Huber 1985):

$$\phi(x) = \sum_{i=1}^p g_i(w_i^T x),$$

where both the directions w_i and functions g_i are learnable parameters, and $w_i^T x$ represents the one-dimensional projection of the input vector x onto the direction defined by w_i . Each $g_i(w_i^T x)$ can be interpreted as a learned feature extracted from the data. Persi Diaconis and Shahshahani (1984) extended this approach using nonlinear functions of linear combinations, establishing the mathematical framework that would later inform the design of multi-layer neural networks.

19.1.1 From Approximation to Representation

While ridge functions and projection pursuit provide powerful approximation tools, they rely on sums of increasing numbers of components to achieve precision. A fundamental question in deep learning theory is whether we can achieve *exact* representation of multivariate functions using a finite number of components, rather than just approximation.

This question addresses a core tension in machine learning: the trade-off between the mathematical efficiency of a representation and its computational feasibility. Modern deep learning succeeds not just because of hardware acceleration (GPUs), but because deep architectures may implicitly leverage efficient representations of high-dimensional functions.

To understand this, we turn to a foundational result that seemingly solves the representation problem entirely: the Kolmogorov Superposition Theorem.

19.2 Kolmogorov Superposition Theorem (KST)

Kolmogorov demonstrated that any real-valued continuous function $f(\mathbf{x})$ on E^n can be represented as a composition of single-variable functions:

$$f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} g_q (\phi_q(x_1, \dots, x_n))$$

where g_q are continuous single-variable functions defined on $\phi_q(E^n)$. Kolmogorov further showed that the ϕ_q functions can be decomposed into sums of single-variable functions:

$$\phi_q(x_1, \dots, x_n) = \sum_{i=1}^n \psi_{q,i}(x_i)$$

The Kolmogorov representation theorem A. Kolmogorov (1956) takes the form:

$$f(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} g_q \left(\sum_{i=1}^n \psi_{q,i}(x_i) \right)$$

The theorem has been refined over time, with inner functions becoming Hölder continuous and Lipschitz continuous, requiring modifications to both outer and inner functions.

The inner functions $\psi_{q,i}$ partition the input space into distinct regions, and the outer function g must be constructed to provide the correct output values across these regions. For each input configuration, the inner functions generate a unique encoding, and g must map this encoding to the appropriate value of $f(x)$. This creates a dictionary-like structure that associates each region with its corresponding output value.

The constructive proof of KST has been refined through several contributions. Sprecher (1965) provided the first explicit construction of the inner functions, though his proof contained technical gaps. Köppen (2000) corrected these errors and provided a complete algorithmic construction. More recently, Actor (2018) and Dembo (2021) proposed computational improvements to the algorithm. Braun and Riedmiller (2009) further enhanced the understanding by providing precise definitions of the shift parameters δ_k and characterizing the topological structure induced by the inner functions.

A fundamental trade-off in KST exists between function smoothness and dimensionality. The inner functions $\psi_{p,q}$ can be chosen from two different function spaces, each offering distinct advantages. The first option is to use functions from $C^1([0, 1])$ —the space of continuously differentiable functions—but this limits the network’s ability to handle higher dimensions effectively. The second option is to relax the smoothness requirement to Hölder continuous functions ($\psi_{p,q} \in \text{Hölder}_\alpha([0, 1])$), which satisfy the inequality $\|\psi(\mathbf{x}) - \psi(\mathbf{y})\| < \|\mathbf{x} - \mathbf{y}\|^\alpha$ for vector-valued inputs. These functions are less smooth, but this “roughness” enables better approximation in higher dimensions.

19.2.1 Kolmogorov-Arnold Networks

A significant development has been the emergence of Kolmogorov-Arnold Networks (KANs). The key innovation of KANs is their use of learnable functions rather than weights on the network edges. This replaces traditional linear weights with univariate functions, typically parametrized by splines, enhancing both representational capacity and interpretability.

A practical connection exists between KST and neural networks: any KAN can be constructed as a 3-layer MLP. Consider a KST in the form of sums of functions, a two layer model:

$$f(x_1, \dots, x_d) = f(x) = (g \circ \psi)(x)$$

Then KAN not only a superposition of functions but also a particular case of a tree of discrete Urysohn operators:

$$U(x_1, \dots, x_d) = \sum_{j=1}^d g_j(x_j)$$

This structure enables fast, scalable algorithms that avoid backpropagation for any GAM model through *projection descent with Kaczmarz schemes*—iterative methods originally developed for solving systems of linear equations that can be adapted for training these networks (Kaczmarz 1937).

19.3 Kolmogorov Generalized Additive Models (K-GAM)

Rather than using learnable functions as network node activations, Polson and Sokolov directly use KST representation. This 2-layer network with non-

differentiable inner function has architecture:

$$f(x_1, \dots, x_d) = \sum_{q=0}^{2d} g_q(z_q)$$

where the inner layer embeds $[0, 1]^d$ to \mathbb{R}^{2d+1} via:

$$z_q = \eta_q(x_1, \dots, x_d) = \sum_{p=1}^d \lambda_p \psi(x_p + qa)$$

Here, $\lambda_p = \sum_{r=1}^{\infty} \gamma^{-(p-1)\beta(r)}$ represents p -adic expansion with $\beta(r) = (n^r - 1)/(n - 1)$ and $\gamma \geq d + 2$, $a = (\gamma(\gamma - 1))^{-1}$.

The Koppen function ψ is defined through a recursive limit:

$$\psi(x) = \lim_{k \rightarrow \infty} \psi_k \left(\sum_{l=1}^k i_l \gamma^{-l} \right)$$

where each $x \in [0, 1]$ has the representation:

$$x = \sum_{l=1}^{\infty} i_l \gamma^{-l} = \lim_{k \rightarrow \infty} \left(\sum_{l=1}^k i_l \gamma^{-l} \right)$$

and ψ_k is defined recursively as:

$$\psi_k = \begin{cases} d, & d \in D_1 \\ \psi_{k-1}(d - i_k \gamma^{-k}) + i_k \gamma^{-\beta_n(k)}, & d \in D_k, k > 1, i_k < \gamma - 1 \\ \frac{1}{2} (\psi_k(d - \gamma^{-k}) + \psi_{k-1}(d + \gamma^{-k})), & d \in D_k, k > 1, i_k = \gamma - 1 \end{cases}$$

The most striking aspect of KST is that it leads to a Generalized Additive Model (GAM) with fixed features that are independent of the target function f . These features, determined by the Koppen function, provide universal topological information about the input space, effectively implementing a k-nearest neighbors structure that is inherent to the representation.

This leads to the following architecture. Any deep learner can be represented as a GAM with feature engineering (topological information) given by features z_k in the hidden layer:

$$y_i = \sum_{k=1}^{2n+1} g(z_k)$$

$$z_k = \sum_{j=1}^n \lambda^k \psi(x_j + \epsilon k) + k$$

where ψ is a single activation function common to all nodes, and g is a single outer function.

One approach replaces each ϕ_j with a single ReLU network g :

$$g(x) = \sum_{k=1}^K \beta_k \text{ReLU}(w_k x + b_k)$$

where K is the number of neurons.

19.4 Space Partitioning

The partitioning of the input space by a deep learner is similar to that performed by decision trees and partition-based models such as CART, MARS, and RandomForests. However, trees are more local in the regions that they use to construct their estimators. Each neuron in a deep learning model corresponds to a manifold that divides the input space. In the case of the ReLU activation function $f(x) = \max(0, x)$, the manifold is simply a hyperplane. The neuron activates when the new observation is on the “right” side of this hyperplane, with the activation magnitude equal to the distance from the boundary. For example, in two dimensions, three neurons with ReLU activation functions will divide the space into seven regions, as shown in Figure 19.1.

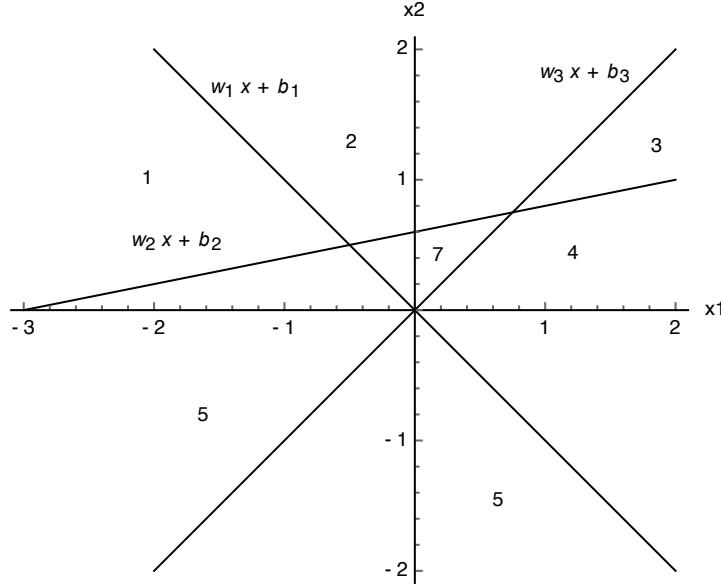


Figure 19.1: Space partition by three ReLU neurons in 2D.

The key difference between tree-based architectures and neural network-based models lies in how hyperplanes are combined. Figure 19.2 compares space decomposition by hyperplanes in tree-based versus neural network architectures. We compare a two-layer neural network (bottom row) with a tree model trained via the CART algorithm (top row). The network architecture used is:

$$\begin{aligned} Y &= \text{softmax}(W^{(0)}Z^{(2)} + b^{(0)}) \\ Z^{(2)} &= \tanh(W^{(2)}Z^{(1)} + b^{(2)}) \\ Z^{(1)} &= \tanh(W^{(1)}X + b^{(1)}) \end{aligned}$$

The weight matrices for simple data are $W^{(1)}, W^{(2)} \in \mathbb{R}^{2 \times 2}$; for circle data $W^{(1)} \in \mathbb{R}^{2 \times 2}$ and $W^{(2)} \in \mathbb{R}^{3 \times 2}$; and for spiral data we have $W^{(1)} \in \mathbb{R}^{2 \times 2}$ and $W^{(2)} \in \mathbb{R}^{4 \times 2}$. An advantage of deep architectures is that the number of hyperplanes grows exponentially with the number of layers.

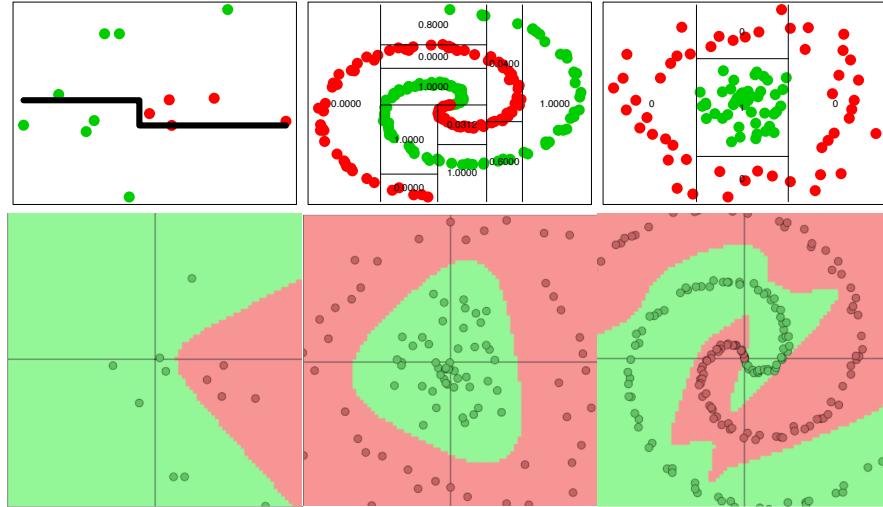


Figure 19.2: Space partition by tree architectures (top row) and deep learning architectures (bottom row) for three different data sets.

19.4.1 Connection to Bayesian Model Averaging

This space partitioning perspective connects naturally to Bayesian ensemble methods. A Bayesian probabilistic approach can optimally weight predictors via model averaging:

$$\hat{Y}(X) = \sum_{r=1}^R w_r \hat{Y}_r(X)$$

where $\hat{Y}_r(X) = E(Y | X, \text{model } r)$ and weights w_r are posterior model probabilities. Amit, Blanchard, and Wilder (2000) demonstrated the success of multiple randomized classifiers (e.g., an ensemble of 100 trees), which reduces error rates significantly compared to single trees by exploiting the weak correlation between diverse classifiers.

19.5 Kernel Smoothing and Interpolation

The theory of kernel methods was developed by Fredholm in the context of integral equations (Fredholm 1903). The idea is to represent a function as a linear combination of basis functions, which are called kernels.

$$f(x) = \int_a^b K(x, x') d\mu(x') \quad \text{where } \mathbf{x} = (x_1, \dots, x_d)$$

Here, the unknown function $f(x)$ is represented as a linear combination of kernels $K(x, x')$ with unknown coefficients $\phi(x')$. The kernels are known, and the coefficients are unknown. The coefficients are found by solving the integral equation. The first work in this area was done by Abel who considered equations of the form above.

Nowadays, we call those equations Volterra integral equations of the first kind. Integral equations typically arise in inverse problems. Their significance extends beyond their historical origins, as kernel methods have become instrumental in addressing one of the fundamental challenges in modern mathematics: the curse of dimensionality.

Nadaraya (1964) and Watson (1964) independently proposed using kernels to estimate the regression function. The idea is to estimate $f(x)$ at a point x by computing a weighted average of the response values y_i at nearby points x_i , with the kernel function defining the weights.

The regression function estimate:

$$\hat{f}(x) = \sum_{i=1}^n y_i K(x, x_i) / \sum_{i=1}^n K(x, x_i),$$

with normalized kernel weights.

Both Nadaraya and Watson considered the symmetric kernel $K(x, x') = K(\|x' - x\|_2)$, where $\|\cdot\|_2$ is the Euclidean norm. The most popular kernel of that sort is the Gaussian kernel:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|_2^2}{2\sigma^2}\right).$$

Alternatively, replacing the 2-norm with inner products: $K(x, x') = \exp(x^T x' / 2\sigma^2)$.

Kernel methods are supported by numerous generalization bounds which often take the form of inequalities that describe the performance limits of kernel-based estimators. A particularly important example is the Bayes risk for k -nearest neighbors (k -NN), which can be expressed in a kernel framework as:

$$\hat{f}(x) = \sum_{i=1}^N w_i y_i \text{ where } w_i := K(x_i, x) / \sum_{i=1}^N K(x_i, x)$$

k -NN classifiers converge to error rates bounded relative to Bayes error rate, with relationships depending on class number. For binary classification, asymptotic k -NN error rate is at most $2R^*(1 - R^*)$ where R^* is Bayes error rate. Cover and Hart proved interpolated k -NN schemes are consistent estimators with performance improving as sample size increases.

19.6 Transformers as Kernel Smoothing

Bahdanau, Cho, and Bengio (2014) proposed kernel smoothing for sequence-to-sequence learning, estimating next-word probability using context vectors—weighted averages of input sequence vectors h_j :

$$c_i = \sum_{j=1}^n \alpha_{ij} h_j,$$

where weights α_{ij} are defined by the kernel function:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^n \exp(e_{ik})}.$$

Instead of using a traditional similarity measure like the 2-norm or inner product, the authors used a neural network to define the energy function $e_{ij} = a(s_{i-1}, h_j)$. This neural network measures the similarity between the last generated element of the output sequence s_{i-1} and j -th element of the input sequence h_j . The resulting context vector is then used to predict the next word in the sequence.

19.6.1 Transformer

Transformers have become the dominant architecture for natural language processing, achieving state-of-the-art results across tasks from machine translation to language modeling and text generation. Originally designed to handle sequential data, the transformer architecture has since been extended to computer vision (Vision Transformers), protein structure prediction (AlphaFold), and speech recognition. Its success stems from a novel self-attention mechanism that efficiently captures long-range dependencies.

The idea to use kernel smoothing for sequence to sequence was called “attention”, or cross-attention, by Bahdanau, Cho, and Bengio (2014). When

used for self-supervised learning, it is called self-attention. When a sequence is mapped to a matrix M , it is called multi-head attention. The concept of self-attention and attention for natural language processing was further developed by Vaswani et al. (2023) who developed a smoothing method that they called the transformer.

The transformer architecture revolves around a series of mathematical concepts and operations:

- **Embeddings:** The input text is converted into vectors using embeddings. Each word (or token) is represented by a unique vector in a high-dimensional space.
- **Positional Encoding:** Since transformers do not have a sense of sequence order (like RNNs do), positional encodings are added to the embeddings to provide information about the position of each word in the sequence.
- **Multi-Head Attention:** The core of the transformer model. It enables the model to focus on different parts of the input sequence simultaneously. The attention mechanism is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

where Q , K , and V are query, key, and value matrices respectively.

- **Query (Q), Key (K), and Value (V) Vectors:** These are derived from the input embeddings. They represent different aspects of the input.
- **Scaled Dot-Product Attention:** The attention mechanism calculates the dot product of the Query with all Keys, scales these values, and then applies a softmax function to determine the weights of the Values.
- **Multiple ‘Heads’:** The model does this in parallel multiple times (multi-head), allowing it to capture different features from different representation subspaces.
- **Layer Normalization and Residual Connections:** After each sub-layer in the encoder and decoder (like multi-head attention or the feed-forward layers), the transformer applies layer normalization and adds the output of the sub-layer to its input (residual connection). This helps in stabilizing the training of deep networks.
- **Feed-Forward Neural Networks:** Each layer in the transformer contains a fully connected feed-forward network applied to each position separately and identically. It is defined as:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

where W_1 , W_2 , b_1 , and b_2 are learnable parameters.

- **Output Linear Layer and Softmax:** The decoder’s final output passes through a linear layer followed by a softmax layer. This layer converts the decoder output into predicted next-token probabilities.

- **Training and Loss Function:** Transformers are often trained using a variant of Cross-Entropy Loss to compare the predicted output with the actual output.
- **Masking:** In the decoder, to prevent future tokens from being used in the prediction, a technique called ‘masking’ is applied.
- **Backpropagation and Optimization:** The model’s parameters are adjusted through backpropagation and optimization algorithms like Adam.

Later, Lin et al. (2017) proposed using similar idea for self-supervised learning, where a sequence of words (sentence) is mapped to a single matrix:

$$M = AH,$$

where H is the matrix representing an input sequence $H = (h_1, \dots, h_n)$ and A is the matrix of weights:

$$A = \text{softmax}(W_2 \tanh(W_1 H^T)).$$

This allows to represent a sequence of words of any length n using a “fixed size” $r \times u$ matrix M , where u is the dimension of a vector that represents an element of a sequence (word embedding) and r is the hyper-parameter that defines the size of the matrix M .

The primary advantage of transformers over recurrent architectures is their parallelizability—all positions in a sequence can be processed simultaneously rather than sequentially. Modern large language models such as BERT (Devlin et al. 2019), GPT (Brown et al. 2020), and T5 (Raffel et al. 2020) are built on transformer architectures. The transformer’s ability to capture long-range dependencies through attention, combined with its scalability to massive datasets and model sizes, has made it the foundation for the current generation of AI systems. See Tsai et al. (2019) for a comprehensive survey of transformer applications beyond NLP.

19.7 Deep Learning as Representation Learning

A fundamental perspective on deep learning is that it automates the process of feature engineering. Formally, we seek to find a predictor $\mathbf{Y} = f(\mathbf{X})$. We can decompose this function into two stages: a data transformation (representation learning) $\phi(\mathbf{X})$ and a predictive model $g(\cdot)$.

$$\mathbf{Y} \sim p(\mathbf{Y} \mid \mathbf{Z}), \quad \mathbf{Z} = \phi(\mathbf{X})$$

Here, \mathbf{Z} represents latent features. The transformation $\phi(\cdot)$ effectively performs dimensionality reduction (or expansion) to uncover a structure where the relationship between \mathbf{Z} and \mathbf{Y} is simpler (often linear).

19.7.1 Dimensionality Expansion vs. Reduction

Classically, statisticians approached complex functions through **dimensionality expansion** (basis expansion). Methods like **Kernel Regression** or **Splines** map the input \mathbf{X} into a higher-dimensional space of features $\phi(\mathbf{X}) = (\phi_1(\mathbf{X}), \dots, \phi_M(\mathbf{X}))$ (e.g., interactions, polynomials) and then fit a linear model. The “Kernel Trick” allows this to be done efficiently without explicit computation of features.

Deep learning, in contrast, often emphasizes **dimensionality reduction** (feature learning). It searches for a low-dimensional manifold embedded in the high-dimensional input space.

19.7.2 Linear Baselines: PCA and PLS

To understand the contribution of deep learning, it is useful to compare it with linear methods for Representation Learning:

1. **Principal Component Analysis (PCA):** An unsupervised method that finds orthogonal directions (eigenvectors of $\mathbf{X}^T \mathbf{X}$) maximizing variance. It creates features $\mathbf{Z}_{PCA} = \mathbf{X}\mathbf{W}$ without knowledge of the target \mathbf{Y} .
2. **Partial Least Squares (PLS):** A supervised method that finds directions maximizing the covariance between \mathbf{X} and \mathbf{Y} . It effectively finds features that are both variable and predictive.

Deep Neural Networks extrapolate this concept to the nonlinear regime. They can be viewed as performing **Nonlinear PLS**. The layers of a network apply successive semi-affine transformations:

$$\mathbf{Z}^{(l)} = \sigma(W^{(l)}\mathbf{Z}^{(l-1)} + b^{(l)})$$

This hierarchical stacking allows the network to learn progressively more abstract representations, discovering nonlinear factors that maximize predictive power, much like a nonlinear generalization of the supervised dimensionality reduction performed by PLS.

19.7.3 Uncertainty Quantification

We can extend this framework to quantify uncertainty. Our probabilistic model is $\mathbf{Y} \mid \mathbf{Z} \sim p(\mathbf{Y} \mid \mathbf{Z})$, where $\mathbf{Z} = g(\mathbf{X})$ is the deep feature extraction. A key result by Brillinger (2012) suggests that for certain input distributions (e.g., Gaussian), the central subspace estimated by methods like PLS or Deep Learning can be consistent.

By treating the top layer as a generalized linear model on learned features \mathbf{Z} , we can leverage standard Bayesian techniques for the final prediction layer while relying on the deep network for feature discovery. This “Last Layer Bayesian” approach provides a practical path to uncertainty quantification in deep learning, enabling the calculation of predictive intervals:

$$\mathbf{Y}_* \sim \int p(\mathbf{Y} \mid \mathbf{Z}_*, \theta) p(\theta \mid \mathcal{D}) d\theta$$

where θ represents the parameters of the output layer.

19.8 Double Descent

Double descent is a phenomenon of over-parameterized statistical models. In this section, we present a view of double descent from a Bayesian perspective. Over-parameterized models such as deep neural networks have an interesting re-descending property in their risk characteristics. This is a recent phenomenon in machine learning and has been the subject of many studies. As the complexity of the model increases, there is a U-shaped region corresponding to the traditional bias-variance trade-off, but then as the number of parameters equals the number of observations and the model becomes one of interpolation, the risk can become infinite and then, in the over-parameterized region, it re-descends—the double descent effect. We show that this has a natural Bayesian interpretation. Moreover, we show that it is not in conflict with the traditional Occam’s razor that Bayesian models possess, in that they tend to prefer simpler models when possible.

Empirically, the double descent effect was initially observed for high-dimensional neural network regression models and the good performance of these models on such tasks as large language models, image processing, and generative AI methods(Nareklishvili, Polson, and Sokolov 2023a). The double descent effect extends the classical bias-variance trade-off curve that shrinkage estimators possess. This phenomenon was first observed in the context of linear regression(Belkin et al. 2019). The authors showed that the test error of the estimator can decrease as the number of parameters increases. Bach (2024) extends these results to stochastic regression models.

Interpolators—estimators that achieve zero training error—were then shown to have attractive properties due to the double descent effect(Hastie et al. 2022). Our goal is to show that Bayesian estimators can also possess a double descent phenomenon. Interpolators such as ReLU neural networks(Nicholas G. Polson, Sokolov, et al. 2017) have increased in popularity with many applications such as traffic flow modeling(Nicholas G. Polson, Sokolov, et al. 2017) and high-frequency trading(M. F. Dixon, Polson, and Sokolov 2019), among many others.

Occam’s razor—the favoring of simpler models over complex ones—is a natural feature of Bayesian methods that are based on the weight of evidence(a.k.a. the marginal likelihood of the data). To do this, they penalize models with higher complexity via a correction term as in the Bayesian Information Criterion (BIC). This seems inconsistent with the double descent phenomenon. We show that this is not the case, as even though Bayesian methods shift the posterior towards lower-complexity models, highly parameterized Bayesian models can also have good risk properties due to the conditional prior of parameters given the model. We illustrate this with an application to neural network models.

Double descent has been studied from a frequentist point of view in Belkin et al. (2019), Bach (2024). The phenomenon of double descent is illustrated in Figure 19.3. The first part of the curve represents the classical U-shaped bias-variance trade-off. The second part demonstrates the double descent phenomenon, where the test error of the estimator can decrease as the model becomes over-parameterized beyond the interpolation threshold. This phenomenon was later observed in the context of deep learning(Nakkiran et al. 2021). The authors showed that the test error of the estimator can decrease as the number of parameters increases.

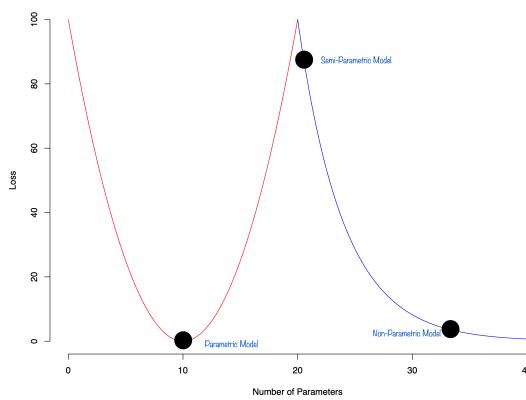


Figure 19.3

Example 19.1 (Double Descent Demonstration using Polynomial Regres-

sion). To illustrate the double descent phenomenon in a concrete setting, we present a detailed example using polynomial regression with Legendre basis functions. This example demonstrates how the test error can exhibit the characteristic U-shaped curve followed by a re-descent as model complexity increases far beyond the interpolation threshold.

Our demonstration uses a one-dimensional regression problem where we attempt to learn a sinusoidal function $f(x) = \sin(5x)$ from a small dataset of only $n = 20$ observations sampled from the interval $[-1, 1]$. We add Gaussian noise with standard deviation $\sigma = 0.3$ to simulate realistic measurement error. The choice of a small sample size is crucial for observing double descent, as it creates a regime where the number of model parameters can substantially exceed the number of observations.

We fit polynomial models of varying degrees $d = 1, 2, \dots, 50$ using Legendre polynomial basis functions. Legendre polynomials provide a numerically stable orthogonal basis that helps avoid the numerical instabilities associated with standard monomial bases in high-degree polynomial fitting. For each degree d , we estimate the coefficients using the Moore-Penrose pseudoinverse, which provides the minimum-norm solution when the system is overdetermined (i.e., when $d > n$).

Figure 19.4 illustrates how model behavior changes dramatically across different polynomial degrees. The four panels show representative cases that capture the key phases of the double descent phenomenon:

- **Degree 1 (Underparameterized):** The linear model is too simple to capture the oscillatory nature of the underlying sine function, resulting in high bias and poor fit to both training and test data.
- **Degree 5 (Classical Optimum):** This represents the sweet spot of the classical bias-variance tradeoff, where the model has sufficient complexity to capture the main features of the sine function without overfitting severely.
- **Degree 20 (Interpolation Threshold):** At this degree, the model has exactly as many parameters as training observations, enabling perfect interpolation of the training data. However, the resulting fit exhibits wild oscillations between data points, leading to poor generalization performance.
- **Degree 50 (Over-parameterized):** Surprisingly, despite having far more parameters than observations, this highly over-parameterized model achieves better test performance than the interpolating model, demonstrating the double descent effect.

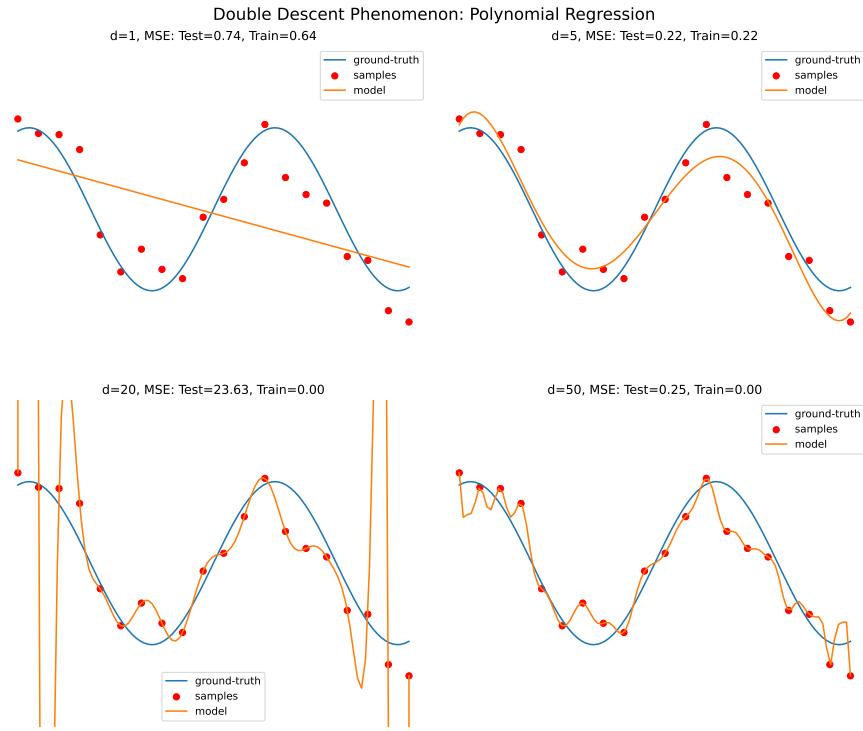


Figure 19.4: Double Descent Phenomenon: Polynomial Regression with Different Degrees

Now, let's plot the MSE curve. We will plot the test error (blue line) and the training error (red line) for different polynomial degrees from 1 to 50.

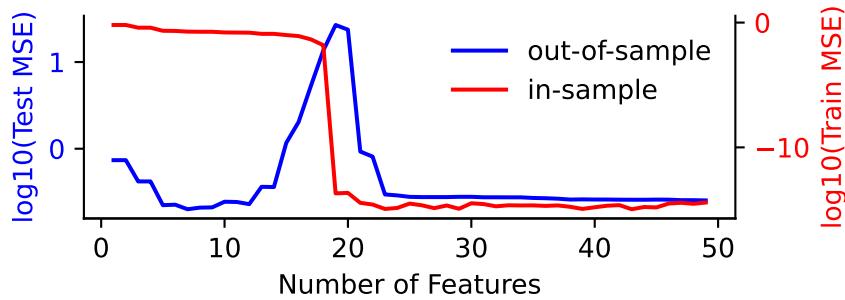


Figure 19.5: Bias-Variance Trade-off: Training and Test MSE vs Model Complexity

The key insight from Figure 19.5 is the characteristic double descent shape in the test error (blue line). The curve exhibits three distinct phases:

1. **Classical Regime:** For low degrees ($d < 5$), increasing model complexity reduces both bias and test error, following the traditional understanding of the bias-variance tradeoff.
2. **Interpolation Crisis:** Around the interpolation threshold ($d \approx n = 20$), test error peaks dramatically as the model begins to perfectly fit the training data while generalizing poorly.
3. **Over-parameterized Regime:** For very high degrees ($d > 30$), test error decreases again, demonstrating that extreme over-parameterization can lead to improved generalization despite the model's ability to memorize the training data.

This behavior challenges the conventional wisdom that more parameters necessarily lead to worse generalization. The double descent phenomenon arises from the implicit regularization effects of minimum-norm solutions in over-parameterized settings. When $d > n$, the pseudoinverse solution corresponds to the minimum ℓ_2 -norm coefficients among all possible interpolating solutions. This implicit bias toward simpler functions can lead to surprisingly good generalization properties.

While this example uses polynomial regression for clarity, the double descent phenomenon has been observed across a wide range of modern machine learning models, including deep neural networks, random forests, and kernel methods. The implications for practice are significant. Given that model selection is time consuming and computationally expensive, this example shows, that instead of spending time to do model selection to find the “sweet spot” model with 5-degree polynomial, we just over-parametrise and get a good model for free!

This example serves as a concrete illustration of how classical statistical intuitions about model complexity may not apply in contemporary machine learning settings, particularly when dealing with over-parameterized models that have become increasingly common in practice.

19.9 Application

19.9.1 Simulated Data

We also apply the K-GAM architecture to a simulated dataset to evaluate its performance on data with known structure and relationships. The dataset

contains 100 observations generated from the following function:

$$\begin{aligned} y &= \mu(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1) \\ \mu(x) &= 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5. \end{aligned}$$

The goal is to predict the function $y(x)$ based on the input x . The dataset is often used as a benchmark dataset for regression algorithms due to its diverse mix of relationships (linear, quadratic, nonlinear, Gaussian random noise) between the input features and the target function.

We use the Koppen function to transform the five-dimensional input into a set of 11 features ($2d + 1$). We then learn the outer function g using a ReLU network. To thoroughly investigate the model's capabilities, we implement two distinct approaches to learning the outer function. The first approach uses different g functions for each feature, following the original KST formulation. This allows each function to specialize in capturing specific patterns, but might be more difficult to train and has more parameters. The second approach uses a single g function for all features, as proposed by Lorentz (1976), providing a more unified and parameter-efficient representation.

The first model with multiple g_i functions has dimensions: $W_i^0 \in \mathbb{R}^{16 \times 1}$ and $W_i^j \in \mathbb{R}^{16 \times 16}$ for $j = 1, \dots, 18$.

The second architecture using single function g for all features maintains similar structure but increases inner layer width from 16 to 200. This increased capacity allows single functions to learn complex patterns, compensating for the constraint versus multiple specialized functions.

19.9.2 Training Rates

Consider nonparametric regression $y_i = f(x_i) + \epsilon_i$ where $x_i = (x_{1i}, \dots, x_{di})$. We estimate $f(x_1, \dots, x_d)$ for $x = (x_1, \dots, x_d) \in [0, 1]^d$. From a classical risk perspective:

$$R(f, \hat{f}_N) = E_{X,Y} (\|f - \hat{f}_N\|^2),$$

where $\|\cdot\|$ denotes $L^2(P_X)$ -norm.

Under standard assumptions, optimal minimax rate $\inf_{\hat{f}} \sup_f R(f, \hat{f}_N) = O_p(N^{-2\beta/(2\beta+d)})$ for β -Holder smooth functions f . This rate depends on dimension d , problematic in high dimensions. Restricting function classes yields better rates independent of d , avoiding the curse of dimensionality. Common approaches include linear superpositions (ridge functions) and projection pursuit models.

Another asymptotic result comes from posterior concentration. Here, \hat{f}_N is a regularized MAP (maximum a posteriori) estimator solving:

$$\hat{f}_N = \arg \min_{\hat{f}_N} \frac{1}{N} \sum_{i=1}^N (y_i - \hat{f}_N(x_i))^2 + \phi(\hat{f}_N)$$

where $\phi(\hat{f})$ is regularization. Under appropriate conditions, posterior distribution $\Pi(f|x, y)$ concentrates around true function at minimax rate (up to $\log N$ factor).

A key result in the deep learning literature provides convergence rates for deep neural networks. Given a training dataset of input-output pairs $(x_i, y_i)_{i=1}^N$ from the model $y = f(x) + \epsilon$ where f is a deep learner (i.e. superposition of functions)

$$f = g_L \circ \dots \circ g_1 \circ g_0$$

where each g_i is a β_i -smooth Hölder function with d_i variables, that is $|g_i(x) - g_i(y)| < |x - y|^{\beta_i}$.

Then, the estimator has optimal rate:

$$O\left(\max_{1 \leq i \leq L} N^{-2\beta^*/(2\beta^*+d_i)}\right) \text{ where } \beta^*_i = \beta_i \prod_{l=i+1}^L \min(\beta_l, 1)$$

This result can be applied to various function classes, including generalized additive models of the form

$$f_0(x) = h \left(\sum_{p=1}^d f_{0,p}(x_p) \right)$$

where $g_0(z) = h(z)$, $g_1(x_1, \dots, x_d) = (f_{01}(x_1), \dots, f_{0d}(x_d))$ and $g_2(y_1, \dots, y_d) = \sum_{i=1}^d y_i$. In this case, $d_1 = d_2 = 1$, and assuming h is Lipschitz, we get an optimal rate of $O(N^{-1/3})$, which is independent of d .

Schmidt-Hieber (2021) show that deep ReLU networks also have optimal rate of $O(N^{-1/3})$ for certain function classes. For 3-times differentiable (e.g. cubic B-splines), Coppejans (2004) finds a rate of $O(N^{-3/7}) = O(N^{-3/(2 \times 3 + 1)})$. Igelnik and Parikh (2003) finds a rate $O(N^{-1})$ for Kolmogorov Spline Networks.

Finally, it's worth noting the relationship between expected risk and empirical risk. The expected risk, R , is typically bounded by the empirical risk plus a term of order $1/\sqrt{N}$:

$$R(y, f^*) \leq \frac{1}{N} \sum_{i=1}^N R(y_i, f^*(x_i)) + O\left(\frac{\|f\|}{\sqrt{N}}\right)$$

where f^* is the minimizer of the expected risk. However, in the case of interpolation, where the model perfectly fits the training data, the empirical risk term becomes zero, leaving only the $O(1/\sqrt{N})$ term.

19.10 Conclusion

This chapter has traced the theoretical foundations of deep learning through the lens of multivariate function approximation. Several key insights emerge from this analysis:

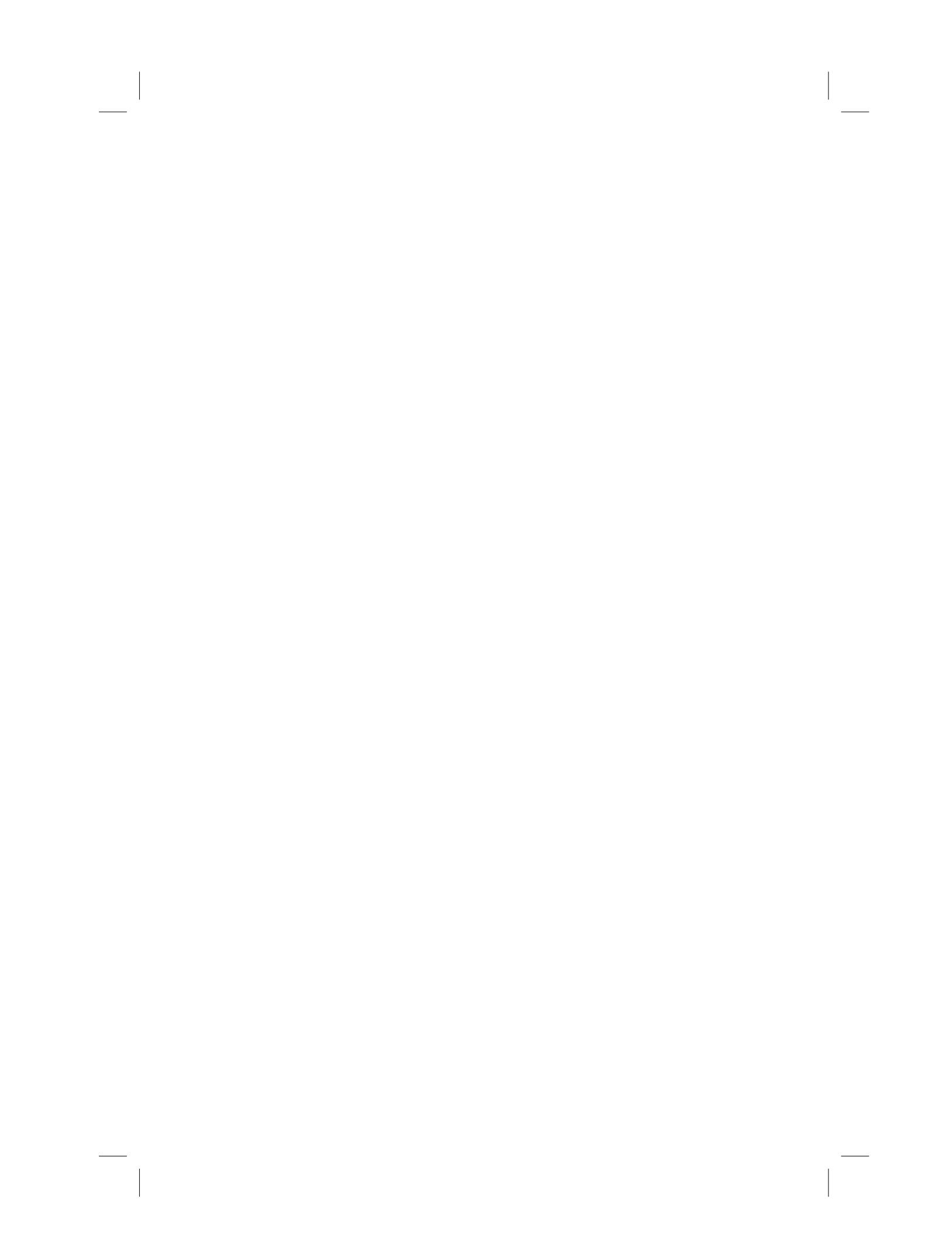
Mathematical Foundations: The Kolmogorov Superposition Theorem provides a rigorous theoretical basis for understanding why neural networks can approximate any continuous function. The theorem shows that any multivariate function can be decomposed into compositions of univariate functions—a principle that directly informs the layered architecture of modern neural networks.

Representation vs. Computation: While KST guarantees the existence of such representations, the inner functions (like the Koppen function) are typically non-smooth and computationally challenging. This highlights a fundamental tension: mathematically elegant representations may not always translate to efficient computation.

Connections Across Methods: We have seen deep connections between seemingly disparate approaches. Transformers can be understood as kernel smoothing methods. Deep networks perform sophisticated space partitioning similar to decision trees. The K-GAM framework reveals that deep learners are essentially generalized additive models with automatically learned features.

Double Descent and Over-parameterization: Perhaps most surprisingly, the double descent phenomenon suggests that classical intuitions about the bias-variance trade-off may not apply to modern over-parameterized models. Networks with more parameters than training examples can still generalize well, challenging conventional wisdom about model complexity.

These theoretical insights not only deepen our understanding of why deep learning works but also suggest new directions for developing more efficient and interpretable architectures. The ongoing research into Kolmogorov-Arnold Networks and related approaches hints at the possibility of achieving the representational power of deep learning with potentially simpler, more interpretable structures.



20

Gradient Descent

“The gradient is the direction of steepest ascent. To minimize, we simply walk the other way.”

How does a neural network with millions of parameters learn the right values? Classical statistical models rely on second-order optimization: least squares for linear regression and Newton-type methods like Broyden-Fletcher-Goldfarb-Shanno (BFGS) for generalized linear models. These algorithms use curvature information—the Hessian matrix—to take efficient steps toward the optimum.

However, second-order optimization algorithms do not scale well to deep learning models. When a model has millions or billions of parameters, computing and storing the Hessian matrix becomes prohibitive from both computational and memory standpoints. Instead, first-order gradient descent methods—which require only gradient information—are the workhorses for training deep neural networks.

This chapter unifies a theme that already appeared earlier: in Chapter 11 we framed estimation as likelihood maximization and showed how the negative log-likelihood becomes a loss. In Chapter 8 we used the same idea computationally to fit Gaussian process hyperparameters by maximizing a marginal likelihood (via `optim`). Here we make the optimization mechanics explicit and reusable.

Given a parametric model $f_\theta(x)$, the problem of parameter estimation (when the likelihood belongs to the exponential family) is an optimization problem:

$$\min_{\theta} l(\theta) := -\frac{1}{n} \sum_{i=1}^n \log p(y_i | x_i, \theta),$$

where l is the negative log-likelihood of a sample (equivalently, $l(\theta) = -\ell(\theta)$ up to scaling, with $\ell(\theta) = \log L(\theta)$ as in Chapter 11), and θ is the vector of parameters. The gradient descent method is an iterative algorithm that starts with an initial guess θ^0 and then updates the parameter vector θ at each iteration t as follows:

$$\theta^{t+1} = \theta^t - \alpha_t \nabla l(\theta^t).$$

Here

$$\nabla l(\theta^t) = \left(\frac{\partial l}{\partial \theta_0}, \dots, \frac{\partial l}{\partial \theta_p} \right),$$

is the gradient of the loss function with respect to the parameters at iteration t .

This algorithm is based on the simple fact that a function value decreases in the direction of the negative gradient, at least locally. This fact follows from the definition of the gradient, which is the vector of partial derivatives of the function with respect to its parameters. The gradient points in the direction of the steepest ascent, and thus the negative gradient points in the direction of the steepest descent. It is easy to see in the figure below, where the function $l(x)$ is shown in blue and the gradient at the point x_0 is shown in red. The negative gradient points in the direction of the steepest descent, which is the direction we want to go to minimize the function. We can easily see it in a one-dimensional case. The derivative is defined as

$$l'(\theta) = \lim_{h \rightarrow 0} \frac{l(\theta + h) - l(\theta)}{h}.$$

The derivative is the slope of the tangent line to the function at the point θ . The derivative is positive, when $l(\theta + h) > l(\theta)$ for small $h > 0$, then the function is increasing at that point, and if it is negative, then the function is decreasing at that point. Thus, we can use the negative derivative to find the direction of the steepest descent.

For the derivative (or gradient in the multivariate case) to be defined, the function $l(\theta)$ must be continuous and *smooth* at the point θ . In practice, this means that the function must not have any sharp corners or discontinuities. For example, $f(\theta) = |\theta|$ has a sharp corner at zero, with a derivative of $+1$ for $\theta > 0$ and -1 for $\theta < 0$. At zero itself, the derivative is undefined—this is called a *non-differentiable point*. Another example is the sign function $f(\theta) = \text{sign}(\theta)$, which has a discontinuity at zero.

This matters for deep learning because the popular ReLU activation function $\max(0, x)$ has a non-differentiable point at zero. Gradient descent—despite operating on these non-smooth loss surfaces—remains the workhorse for training deep neural networks. In practice, this works because: (1) the probability of landing exactly at a non-differentiable point is zero for continuous distributions, and (2) subgradients provide valid descent directions even at kinks.

The second component is the learning rate α_t , a positive scalar controlling the step size. If the learning rate is too small, convergence is slow; if too large, the algorithm may diverge or oscillate. Figure 20.1 shows this effect: when $\alpha = 0.5$, the algorithm converges in one step from $\theta_0 = 2$; when $\alpha = 0.25$, it takes two steps; when $\alpha = 1$, it overshoots the minimum.

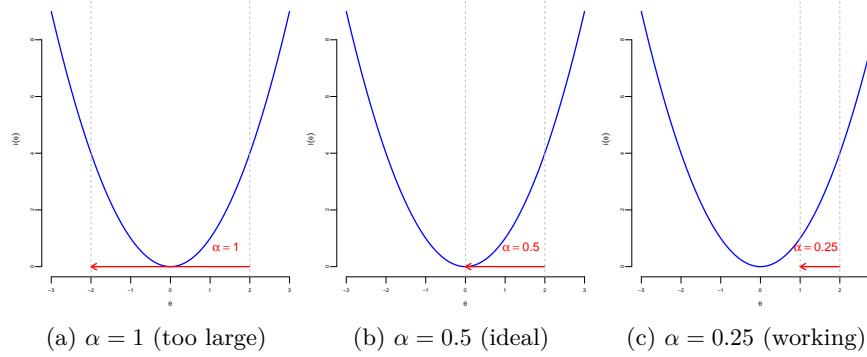


Figure 20.1: Effect of Learning Rates on Gradient Descent

In practice, the learning rate is often set to a small value, such as 0.01 or 0.001, and then adjusted during the training process. There are several techniques for adjusting the learning rate, such as learning rate decay, where the learning rate is reduced by a factor at each epoch, or adaptive learning rates, where the learning rate is adjusted based on the gradient of the loss function.

20.1 Deep Learning and Least Squares

The deep learning model approximates the relation between inputs x and outputs y using a non-linear function $f_\theta(x)$, where θ is a vector of parameters. The goal is to find the optimal value of θ that minimizes the negative log-likelihood (loss function), given a training data set $D = \{x_i, y_i\}_{i=1}^n$. The loss function is a measure of discrepancy between the true value of y and the predicted value $f_\theta(x)$

$$l(\theta) = - \sum_{i=1}^n \log p(y_i|x_i, \theta),$$

where $p(y_i|x_i, \theta)$ is the conditional likelihood of y_i given x_i and θ . In the case of regression, we have

$$y_i = f_\theta(x_i) + \epsilon, \quad \epsilon \sim N(0, \sigma^2),$$

The loss function is then (dropping the constant term $-n \log(\sigma\sqrt{2\pi})$):

$$l(\theta) = - \sum_{i=1}^n \log p(y_i|x_i, \theta) \propto \sum_{i=1}^n (y_i - f_\theta(x_i))^2,$$

Now, let's demonstrate the gradient descent algorithm on a simple example of linear regression using the `iris` dataset.

Example 20.1 (Linear Regression). Linear regression can be viewed as a degenerate neural network: a single layer with no hidden units and an identity activation function. The insight of deep learning is that using *deep and narrow* architectures—multiple layers with fewer units per layer—enables learning complex, hierarchical representations.

Let's fit a linear regression model to the `iris` dataset using gradient descent. We use $y = \text{Petal.Length}$ as the response variable and $x = \text{Petal.Width}$ as the predictor. The model is

$$y_i = \theta_0 + \theta_1 x_i + \epsilon_i,$$

or in matrix form:

$$y = X\theta + \epsilon,$$

where $\epsilon_i \sim N(0, \sigma^2)$, $X = [1 \ x]$ is the design matrix with first column being all ones.

The negative log-likelihood function for the linear regression model is:

$$l(\theta) = \sum_{i=1}^n (y_i - \theta_0 - \theta_1 x_i)^2.$$

The gradient is then:

$$\nabla l(\theta) = -2 \sum_{i=1}^n (y_i - \theta_0 - \theta_1 x_i) \begin{pmatrix} 1 \\ x_i \end{pmatrix}.$$

In matrix form, we have:

$$\nabla l(\theta) = -2X^T(y - X\theta).$$

First, let's load the data and prepare the design matrix. We will use the `Petal.Length` as a response variable and `Petal.Width` as a predictor.

```
data(iris)
y = iris$Petal.Length
```

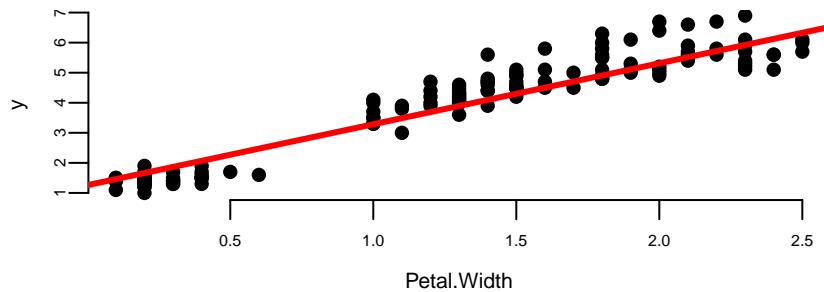
Let's implement the gradient descent algorithm to estimate the parameters of the linear regression model. We will use a small learning rate and a large number of iterations to ensure convergence.

Our gradient descent minimization algorithm finds the following coefficients

Intercept (θ_0)	Petal.Width (θ_1)
1.3	2

Let's plot the data and model estimated using gradient descent

```
plot(x[,2],y,pch=16, xlab="Petal.Width")
abline(a=theta[1], b=theta[2], lwd=3, col="red")
```



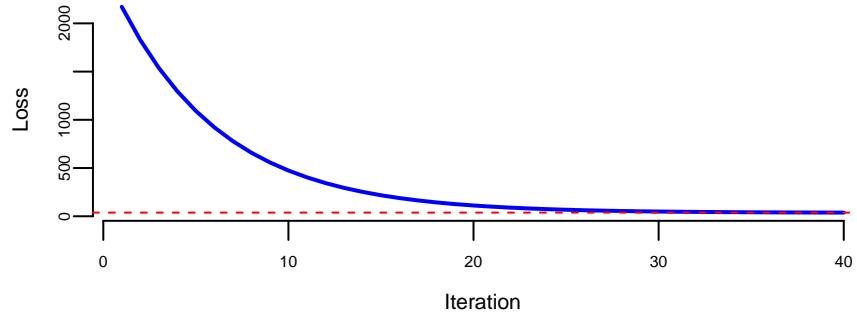
Let's compare it to the standard estimation algorithm

```
m = lm(Petal.Length~Petal.Width, data=iris)
```

(Intercept)	Petal.Width
1.1	2.2

The values found by gradient descent are very close to the ones found by the standard OLS algorithm.

The standard way to monitor the convergence of the gradient descent algorithm is to plot the value of the loss function at each iteration.



We can see that the loss smoothly decays to the minimum value. This is because our loss function is convex (quadratic function of the parameters θ), meaning that it has a single global minimum and no local minima and our learning rate of 0.0001 is small enough to ensure that the algorithm does not overshoot the minimum. However, this is often not the case for more complex models, such as deep neural networks. In those cases, the loss function may oscillate between multiple local minima and saddle points, which can make the convergence of the gradient descent algorithm difficult. In practice, we often use a larger learning rate and a modified version of the gradient descent algorithm, such as Nesterov accelerated gradient descent (NAG) or Adam, to help with the convergence.

Example 20.2 (Logistic Regression). Logistic regression is a generalized linear model (GLM) with a logit link function, defined as:

$$\log\left(\frac{p}{1-p}\right) = \theta_0 + \theta_1 x_1 + \dots + \theta_p x_p,$$

where p is the probability of the positive class. The negative log-likelihood function for logistic regression is a cross-entropy loss

$$l(\theta) = -\sum_{i=1}^n [y_i \log p_i + (1 - y_i) \log(1 - p_i)],$$

where $p_i = 1 / (1 + \exp(-\theta_0 - \theta_1 x_{i1} - \dots - \theta_p x_{ip}))$. The derivative of the negative log-likelihood function is

$$\nabla l(\theta) = -\sum_{i=1}^n [y_i - p_i] \begin{pmatrix} 1 \\ x_{i1} \\ \vdots \\ x_{ip} \end{pmatrix}.$$

In matrix notation, we have

$$\nabla l(\theta) = -X^T(y - p).$$

Let's implement the gradient descent algorithm now.

```

y = ifelse(iris$Species=="setosa",1,0)
x = cbind(rep(1,150),iris$Sepal.Length)
lrgd = function(x,y, alpha, n_iter) {
  theta <- matrix(c(0, 0), nrow = 2, ncol = 1)
  for (i in 1:n_iter) {
    # compute gradient
    p = 1/(1+exp(-x %*% theta))
    grad <- -t(x) %*% (y - p)
    # update theta
    theta <- theta - alpha * grad
  }
  return(theta)
}
theta = lrgd(x,y,0.005,20000)

```

The gradient descent parameters are

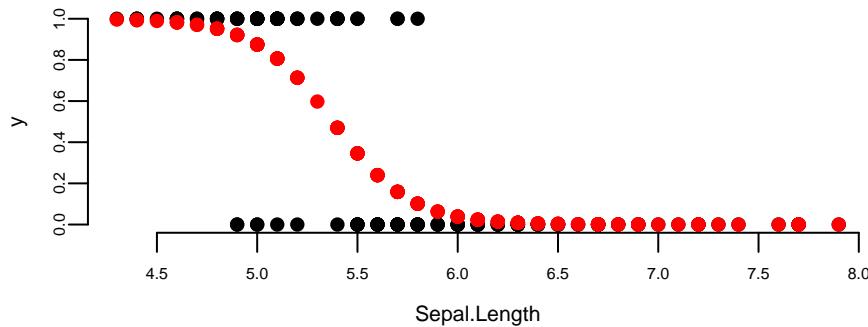
Intercept (θ_0)	Sepal.Length (θ_1)
28	-5.2

And the plot is

```

plot(x[,2],y,pch=16, xlab="Sepal.Length")
lines(x[,2],p,type='p', pch=16,col="red")

```



Let's compare it to the standard estimation algorithm

term	estimate	std.error	statistic	p.value
x1	27.8	4.83	5.8	0
x2	-5.2	0.89	-5.8	0

20.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a variant of the gradient descent algorithm. The main difference is that instead of computing the gradient over the entire dataset, SGD computes the gradient over a randomly selected subset of the data. This allows SGD to be applied to estimate models when the dataset is too large to fit into memory, which is often the case with deep learning models. The SGD algorithm replaces the gradient of the negative log-likelihood function with the gradient computed over a randomly selected subset of the data:

$$\nabla l(\theta) \approx \frac{1}{|B|} \sum_{i \in B} \nabla l(y_i, f(x_i, \theta)),$$

where $B \subseteq \{1, 2, \dots, n\}$ is a *batch* (random subset) sampled from the dataset, and $|B|$ denotes the *batch size* (number of samples in the batch). This method can be interpreted as gradient descent using noisy gradients, which are typically called *mini-batch gradients*.

SGD is based on the idea of stochastic approximation introduced by Robbins and Monro (1951). Stochastic approximation simply replaces the expected gradient with its Monte Carlo approximation.

In a small mini-batch regime, when $|B| \ll n$ and typically $|B| \in \{32, 64, \dots, 1024\}$, it was shown that SGD converges faster than standard gradient descent, does converge to minimizers of strongly convex functions (the negative log-likelihood from the exponential family is strongly convex) (Bottou, Curtis, and Nocedal 2018), and is more robust to noise in the data (Hardt, Recht, and Singer 2016). Further, SGD can avoid saddle points, which is often an issue with deep learning loss functions. In the case of multiple minima, SGD can find a good solution (Keskar et al. 2016; LeCun et al. 2002), meaning that out-of-sample performance is often worse when trained with large-batch methods compared to small-batch methods.

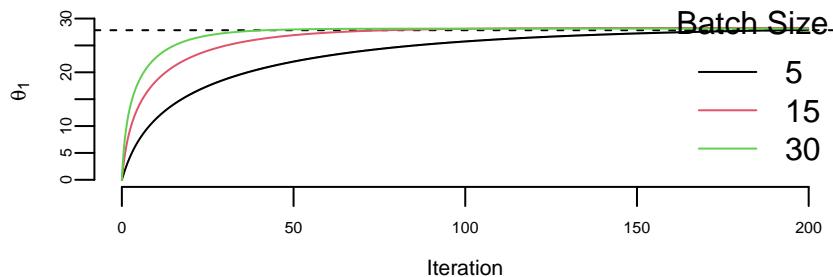
Now, we implement SGD for logistic regression and compare performance for different batch sizes

Now run our SGD algorithm with different batch sizes.

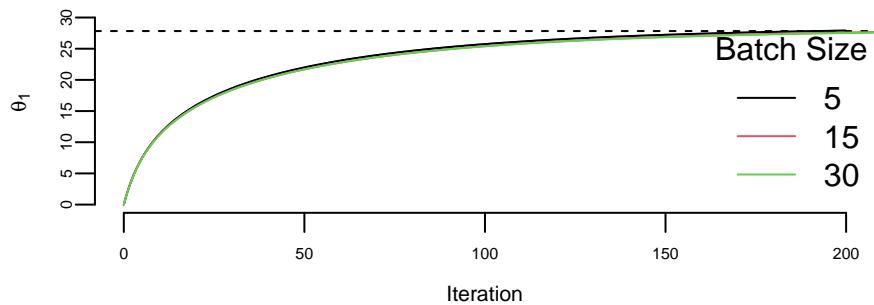
```

set.seed(92) # kuzy
ind = sample(150)
y = ifelse(iris$Species=="setosa",1,0)[ind] # shuffle data
x = cbind(rep(1,150),iris$Sepal.Length)[ind,] # shuffle data
nit=200000
lr = 0.01
th1 = lrgd_minibatch(x,y,lr,nit,5)
th2 = lrgd_minibatch(x,y,lr,nit,15)
th3 = lrgd_minibatch(x,y,lr,nit,30)

```



We run it with 200000 iterations and the learning rate of 0.01 and plot the values of θ_1 every 1000 iteration. There are a couple of important points we need to highlight when using SGD. First, we shuffle the data before using it. The reason is that if the data is sorted in any way (e.g., by date or by value of one of the inputs), then data within batches can be highly correlated, which reduces the convergence speed. Shuffling helps avoid this issue, but it also implicitly treats the dataset as exchangeable; for time series and streaming data, shuffling is typically inappropriate (see Chapter 15). Second, the larger the batch size, the smaller the number of iterations required for convergence, which is something we would expect. However, in this specific example, from the computational point of view, the batch size does not change the number of calculations required overall. Let's look at the same plot, but scale the x-axis according to the amount of computations



There are several important considerations about choosing the batch size for SGD.

- The larger the batch size, the more memory is required to store the data.
- Parallelization is more efficient with larger batch sizes. Modern hardware supports parallelization of matrix operations, which is the main operation in SGD. The larger the batch size, the more efficient the parallelization is. Usually there is a sweet spot $|B|$ for the batch size, which is the largest batch size that can fit into memory or be parallelized. This means it takes the same amount of time to compute an SGD step for batch size 1 and B .
- Third, the larger the batch size, the less noise in the gradient. This means that the larger the batch size, the more accurate the gradient is. However, it was empirically shown that in many applications we should prefer noisier gradients (small batches) to obtain high-quality solutions when the objective function (negative log-likelihood) is non-convex (Keskar et al. 2016).

20.3 Automatic Differentiation (Backpropagation)

Gradient descent requires computing gradients at each iteration. But how do we efficiently compute gradients for complex compositional functions like deep neural networks? The answer lies in *automatic differentiation*—a technique whose development is intertwined with the history of neural networks themselves.

Original neural networks (Perceptrons) proposed by Rosenblatt (1958) were single-layer feedforward networks with a sigmoid activation function in the output layer. They were trained using the so-called *delta rule*.

The delta rule, introduced by Rosenblatt (1958), provides a simple learning algorithm for single-layer networks. For a single-layer perceptron with output

$\hat{y} = \sigma(w^T x + b)$, where σ is an activation function, the delta rule updates weights as:

$$w^{t+1} = w^t - \alpha(\hat{y} - y)\sigma'(w^T x + b)x$$

where α is the learning rate and y is the target output. This rule is intuitive: when the prediction \hat{y} differs from the target y , we adjust the weights proportionally to the input x and the derivative of the activation function.

However, the delta rule encounters a fundamental limitation with multi-layer networks. Consider a network with hidden layers: $h = \sigma_1(W_1 x + b_1)$ and $\hat{y} = \sigma_2(W_2 h + b_2)$. To update weights W_1 in the first layer, we need to compute $\frac{\partial L}{\partial W_1}$, where L is the loss function. By the chain rule:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h} \cdot \frac{\partial h}{\partial W_1}$$

The delta rule could not efficiently compute this gradient through multiple layers, as it lacked a systematic method for propagating error signals backward through the network. This meant that while single-layer networks could be trained effectively, there was no practical way to train deeper architectures that could learn more complex representations.

The inability to train multi-layer networks led to the first AI winter in the late 1960s and 1970s. Minsky and Papert (1969) demonstrated that single-layer perceptrons could not solve non-linearly separable problems like XOR, and without a method to train deeper architectures, neural network research largely stagnated.

Unknown to most of the neural network community at the time, the solution to training multi-layer networks through systematic application of the chain rule was being developed independently by several researchers. In the Soviet Union, Galushkin (1973) developed methods for training multi-layer networks using gradient-based approaches, though this work remained largely unknown in the West due to the Cold War.

In the Western literature, the first clear formulation of backpropagation came from P. Werbos (1974), who in his 1974 PhD thesis at Harvard University introduced the idea of using the chain rule to efficiently compute gradients in multi-layer networks. Werbos explicitly showed how to propagate derivatives backward through a network by decomposing complex functions into simpler components. He refined these ideas further in P. J. Werbos (1982), demonstrating how reverse-mode automatic differentiation could be applied to train neural networks efficiently.

Interestingly, the same mathematical principle—systematic application of the chain rule for computing derivatives—was being applied in a different context around the same time. Nelder and Wedderburn (1972) introduced Generalized Linear Models (GLMs) and developed an iteratively reweighted least squares

(IRLS) algorithm that explicitly used the chain rule to compute gradients for maximum likelihood estimation.

To understand their contribution, consider a GLM where the response variable y_i follows a distribution from the exponential family with mean μ_i . The model relates the mean to the predictors through a link function $g(\cdot)$:

$$g(\mu_i) = \eta_i = x_i^T \theta$$

where x_i is the vector of predictors for observation i , θ is the parameter vector, and η_i is the linear predictor. For example, in logistic regression (which we saw in Example 20.2), we have $g(\mu) = \log(\mu/(1 - \mu))$ (the logit link).

The key innovation of Nelder and Wedderburn was recognizing that computing the gradient of the log-likelihood with respect to θ requires systematic application of the chain rule. Specifically, for observation i , the gradient decomposes as:

$$\frac{\partial \log L}{\partial \theta_j} = \frac{\partial \log L}{\partial \mu_i} \cdot \frac{\partial \mu_i}{\partial \eta_i} \cdot \frac{\partial \eta_i}{\partial \theta_j}$$

The first term, $\partial \log L / \partial \mu_i$, depends on the specific distribution (e.g., Gaussian, Bernoulli, Poisson). For exponential family distributions, this takes the form $(y_i - \mu_i) / V(\mu_i)$, where $V(\mu_i)$ is the variance function. The second term, $\partial \mu_i / \partial \eta_i$, is the derivative of the inverse link function. The third term is simply x_{ij} , the j -th predictor value.

Combining these using the chain rule, they derived the gradient:

$$\frac{\partial \log L}{\partial \theta_j} = \sum_{i=1}^n \frac{(y_i - \mu_i)}{V(\mu_i)} \cdot \frac{\partial \mu_i}{\partial \eta_i} \cdot x_{ij}$$

For the Newton-Raphson optimization method (also known as Fisher scoring when using expected second derivatives), they needed to compute the Hessian matrix. Applying the chain rule again to the gradient yields the Fisher information matrix with elements:

$$\mathcal{I}_{jk} = \sum_{i=1}^n w_i x_{ij} x_{ik}$$

where the weight $w_i = \left(\frac{\partial \mu_i}{\partial \eta_i} \right)^2 / V(\mu_i)$ is itself computed by the chain rule. This leads to the iteratively reweighted least squares algorithm, where at each iteration t we solve:

$$\theta^{t+1} = \theta^t + \mathcal{I}(\theta^t)^{-1} \nabla \log L(\theta^t)$$

The success of GLMs demonstrated that chain rule-based gradient computation was practically valuable for fitting sophisticated models. Nelder and

Wedderburn published their GLM framework in 1972, just two years before Werbos's PhD thesis on backpropagation—both independently discovered that systematic chain rule application could efficiently compute gradients in compositional models. Despite these parallel discoveries, backpropagation did not enter mainstream neural network research until Rumelhart, Hinton, and Williams (1986) popularized it and coined the term *backpropagation*.

To calculate the value of the gradient vector at each step of the optimization process, gradient descent algorithms require calculations of derivatives. In general, there are three different ways to calculate those derivatives. First, we can do it by hand. This is slow and error-prone. Second is numerical differentiation, when a gradient is approximated by a finite difference; for some small h , calculate $f'(x) \approx (f(x+h) - f(x))/h$, which requires two function evaluations. This approach is not backward stable (Griewank, Kulshreshtha, and Walther 2012), meaning that for a small perturbation in input value x , the calculated derivative is not the correct one. Lastly, we can use symbolic differentiation or automatic differentiation (AD) which has been used for decades in computer algebra systems such as **Mathematica** or **Maple**. Symbolic differentiation uses a tree form representation of a function and applies the chain rule to the tree to calculate the symbolic derivative of a given function.

The advantage of symbolic calculations is that the analytical representation of the derivative is available for further analysis, for example, when derivative calculation is an intermediate step of the analysis. A third way to calculate a derivative is to use automatic differentiation (AD). Similar to symbolic differentiation, AD recursively applies the chain rule and calculates the exact value of the derivative, thus avoiding the problem of numerical instability. The difference between AD and symbolic differentiation is that AD provides the value of the derivative evaluated at a specific point, rather than an analytical representation of the derivative.

AD does not require analytical specification and can be applied to a function defined by a sequence of algebraic manipulations, logical and transcendent functions applied to input variables and specified in computer code. AD can differentiate complex functions which involve IF statements and loops, and AD can be implemented using either forward or backward mode. Consider an example of calculating a derivative of the logistic function

$$f(x) = \frac{1}{1 + e^{-(Wx+b)}}.$$

Which is implemented as follows:

Given the function $f(x) = \frac{1}{1+e^{-(Wx+b)}} = v_3$, the derivative with respect to w is given by the chain rule. Recall that for the sigmoid function $\sigma(z) = 1/(1+e^{-z})$, we have $\sigma'(z) = \sigma(z)(1 - \sigma(z))$. Therefore:

$$\frac{\partial f}{\partial w} = \frac{\partial v_3}{\partial v_2} \cdot \frac{\partial v_2}{\partial v_1} \cdot \frac{\partial v_1}{\partial w} = v_3(1 - v_3) \cdot 1 \cdot x$$

$$\frac{\partial f}{\partial b} = \frac{\partial v_3}{\partial v_2} \cdot \frac{\partial v_2}{\partial b} = v_3(1 - v_3) \cdot 1$$

In the forward mode, an auxiliary variable, called a dual number, will be added to each line of the code to track the value of the derivative associated with this line. In our example, if we set $x=2$, $w=3$, $b=5$, we get the calculations given in the table below.

Function calculations	Derivative calculations
1. $v1 = w*x = 6$	1. $dv1 = w = 3$ (derivative of $v1$ with respect to x)
2. $v2 = v1 + b = 11$	2. $dv2 = dv1 = 3$ (derivative of $v2$ with respect to x)
3. $v3 = 1/(1+exp(-v2)) = 0.99$	3. $dv3 = eps2*exp(-v2)/(1+exp(-v2))^2 = 5e-05$

Variables $dv1, dv2, dv3$ correspond to partial (local) derivatives of each intermediate variable $v1, v2, v3$ with respect to x , and are called dual variables. Tracking for dual variables can either be implemented using source code modification tools that add new code for calculating the dual numbers or via operator overloading.

The reverse AD also applies the chain rule recursively but starts from the outer function, as shown in the table below.

Function calculations	Derivative calculations
1. $v1 = w*x = 6$	4. $dv1dx = w; dv1 = dv2*dv1dx = 3*1.3e-05=5e-05$
2. $v2 = v1 + b = 11$	3. $dv2dv1 = 1; dv2 = dv3*dv2dv1 = 1.3e-05$
3. $v3 = 1/(1+exp(-v2)) = 0.99$	2. $dv3dv2 = exp(-v2)/(1+exp(-v2))^2;$
4. $v4 = v3$	1. $dv4=1$

The key difference between forward and reverse mode is the direction of computation:

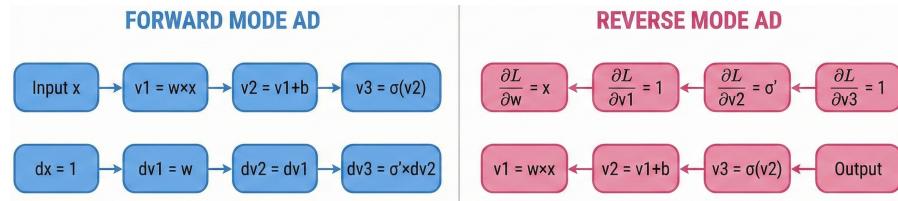


Figure 20.2: Comparison of Forward and Reverse Mode Automatic Differentiation

- **Forward mode:** Propagates derivatives *with* the computation, from inputs to outputs. Efficient when number of inputs < number of outputs.
- **Reverse mode:** Propagates derivatives *against* the computation, from outputs to inputs. Efficient when number of outputs < number of inputs.

For neural networks with millions of parameters (inputs to the loss function) but a single scalar loss (one output), reverse mode is dramatically more efficient—this is why backpropagation uses reverse mode AD.

Figure 20.3 shows a tree representation of the composition of affine and sigmoid functions (the first layer of our neural network). The numbers in the nodes are the values of the variables at a specific point, and the arrows show the flow of data. During the forward pass (solid arrows), the data flows from inputs (x , w , b) to the final output v_3 and all intermediate variables are computed. During the backward pass (dashed arrows), the gradient values flow from the output back to the inputs by applying the chain rule to each variable.

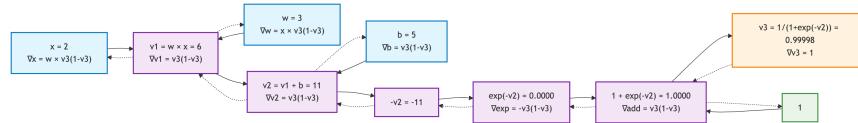


Figure 20.3: Computational Graph for Sigmoid Function

For DL, derivatives are calculated by applying the reverse AD algorithm to a model which is defined as a superposition of functions. A model is defined either using a general-purpose language as it is done in PyTorch or through a sequence of function calls defined by framework libraries (e.g., in TensorFlow). Forward AD algorithms calculate the derivative with respect to a single input variable, but reverse AD produces derivatives with respect to all intermediate variables. For models with many parameters, it is much more computationally feasible to perform the reverse AD.

In the context of neural networks, the reverse AD algorithm is called *backpropagation* and was popularized in AI by Rumelhart, Hinton, and Williams (1986).

According to Schmidhuber (2015), the first version of what we call today backpropagation was published in 1970 in a master's thesis Linnainmaa (1970) and was closely related to the work of Ostrovskii, Volin, and Borisov (1971). However, similar techniques rooted in Pontryagin's maximum principle were discussed in the context of multi-stage control problems (Arthur E. Bryson 1961; Arthur E. Bryson and Ho 1969). Dreyfus (1962) applies backpropagation to calculate the first-order derivative of a return function to numerically solve a variational problem. Later Dreyfus (1973) used backpropagation to derive an efficient algorithm to solve a minimization problem. The first neural network-specific version of backpropagation was proposed in P. Werbos (1974) and an efficient backpropagation algorithm was discussed in P. J. Werbos (1982).

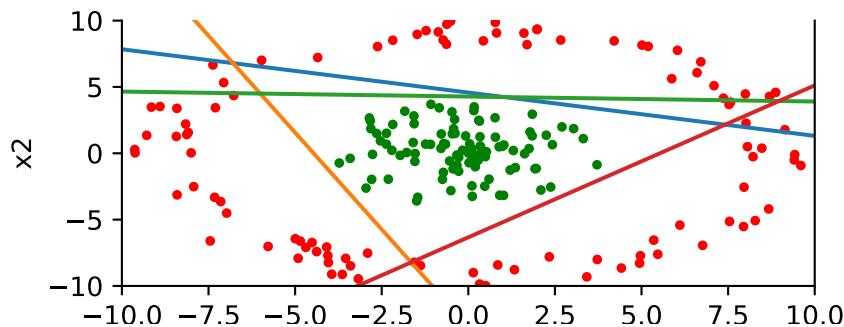
Modern deep learning frameworks fully automate the process of finding derivatives using AD algorithms. For example, PyTorch relies on the `autograd` library which automatically finds gradients using the back-propagation algorithm. Here is a small code example using the `autograd` library in `jax`.

The code below implements a simple neural network with one hidden layer and uses back-propagation to find the gradient of the negative log-likelihood function. The model is trained on a synthetic dataset generated from a circle. The code uses the `jax` library, which is a high-performance numerical computing library that supports automatic differentiation.

```
## 1.0
```

Note that inside the gradient descent function `gd_step` we use the `grad` function to calculate the gradient of the negative log-likelihood function with respect to the weights and biases. The `jit` decorator is used to compile the function for faster execution. The `gd_step` function updates the weights and biases using the calculated gradients and a learning rate of 0.02.

Finally, we can plot the decision boundary of the trained model



We can see that the model has learned to separate the two classes. Two of the lines are almost parallel, which is expected. We could have gotten away with using only three hidden units instead of four.

20.4 Advanced Optimization Methods

While the basic SGD algorithm introduced earlier works well for many problems, practitioners have developed numerous enhancements to improve convergence speed and stability. This section surveys the most important variants used in modern deep learning.

20.4.1 Momentum Methods

One limitation of vanilla SGD is that the descent in the loss function is not guaranteed at every iteration, and progress can be slow when gradients are noisy. To address this, *momentum-based* methods add memory to the search process by combining new gradient information with previous search directions (Nesterov 1983). Empirically, momentum-based methods have been shown to have better convergence for deep learning networks (Sutskever et al. 2013). The gradient only influences changes in the velocity of the update, which then updates the variable:

$$\begin{aligned} v^{k+1} &= \mu v^k - t_k g((W, b)^k) \\ (W, b)^{k+1} &= (W, b)^k + v^k \end{aligned}$$

The hyperparameter μ controls the damping effect on the rate of update of the variables. The physical analogy is the reduction in kinetic energy that allows to “slow down” the movements at the minima. This parameter can also be chosen empirically using cross-validation.

Nesterov’s momentum method (a.k.a. Nesterov acceleration) calculates the gradient at the point predicted by the momentum. One can view this as a look-ahead strategy with the updating scheme

$$\begin{aligned} v^{k+1} &= \mu v^k - t_k g((W, b)^k + v^k) \\ (W, b)^{k+1} &= (W, b)^k + v^k \end{aligned}$$

20.4.2 Adaptive Learning Rate Methods

Another popular modification is **AdaGrad** (Duchi, Hazan, and Singer 2011), which adaptively scales the learning rate for each parameter based on the history of gradients:

$$\begin{aligned} c^{k+1} &= c^k + g(\theta^k)^2 \\ \theta^{k+1} &= \theta^k - \frac{t_k}{\sqrt{c^{k+1}} + \epsilon} g(\theta^k) \end{aligned}$$

where ϵ is a small constant (e.g., $\epsilon = 10^{-8}$) to prevent division by zero. **RMSprop** (Hinton, Srivastava, and Swersky 2012) extends AdaGrad by using an exponentially weighted moving average of squared gradients:

$$\begin{aligned} c^{k+1} &= \rho c^k + (1 - \rho)g(\theta^k)^2 \\ \theta^{k+1} &= \theta^k - \frac{t_k}{\sqrt{c^{k+1}} + \epsilon} g(\theta^k) \end{aligned}$$

The **Adam** method (Kingma and Ba 2014) combines RMSprop with momentum by maintaining exponentially weighted averages of both the gradient (first moment) and the squared gradient (second moment):

$$\begin{aligned} m^{k+1} &= \beta_1 m^k + (1 - \beta_1)g(\theta^k) \\ c^{k+1} &= \beta_2 c^k + (1 - \beta_2)g(\theta^k)^2 \\ \theta^{k+1} &= \theta^k - \frac{t_k}{\sqrt{\hat{c}^{k+1}} + \epsilon} \hat{m}^{k+1} \end{aligned}$$

where \hat{m}^{k+1} and \hat{c}^{k+1} are bias-corrected estimates. The typical hyperparameter values are $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

The following table summarizes the key optimization methods:

Method	Key Idea	Update Rule
SGD	Noisy gradient estimates	$\theta^{k+1} = \theta^k - t_k g^k$
Momentum	Accumulate gradient direction	$v^{k+1} = \mu v^k + g^k;$ $\theta^{k+1} = \theta^k - t_k v^{k+1}$
Nesterov	Look-ahead gradient	$v^{k+1} = \mu v^k + g(\theta^k + \mu v^k)$
AdaGrad	Per-parameter learning rates	Scale by $1/\sqrt{\sum g^2}$
RMSprop	Exponential moving average	Scale by $1/\sqrt{\text{EMA}(g^2)}$

Method	Key Idea	Update Rule
Adam	Momentum + adaptive rates	Combines first and second moment estimates

20.4.3 Second-Order Methods

Second-order methods solve the optimization problem by solving a system of nonlinear equations $\nabla f(\theta) = 0$ using Newton's method:

$$\theta^+ = \theta - \{\nabla^2 f(\theta)\}^{-1} \nabla f(\theta)$$

We can see that SGD effectively approximates the inverse Hessian $\{\nabla^2 f(\theta)\}^{-1}$ by $t_k I$, where I is the identity matrix. The advantages of second-order methods include much faster convergence rates and insensitivity to the conditioning of the problem. However, in practice, second-order methods are rarely used for deep learning applications (Dean et al. 2012). The major disadvantage is the inability to train models using mini-batches as SGD does. Since typical deep learning models rely on large-scale datasets, computing and storing the Hessian becomes memory and computationally prohibitive.

20.5 Why Robbins-Monro?

The Robbins-Monro algorithm was introduced in their seminal 1951 paper “A Stochastic Approximation Method” Robbins and Monro (1951). The paper addressed the problem of finding the root of a function when only noisy observations are available.

Consider a function $M(\theta)$ where we want to find θ^* such that $M(\theta^*) = \alpha$ for some target value α . In the original formulation, $M(\theta)$ represents the expected value of some random variable $Y(\theta)$:

$$M(\theta) = \mathbb{E}[Y(\theta)] = \alpha$$

The key insight is that we can only observe noisy realizations $y(\theta)$ where:

$$y(\theta) = M(\theta) + \epsilon(\theta)$$

where $\epsilon(\theta)$ is a zero-mean random error term.

The Robbins-Monro algorithm iteratively updates the estimate θ_n using:

$$\theta_{n+1} = \theta_n - a_n(y(\theta_n) - \alpha)$$

where a_n is a sequence of positive step sizes that must satisfy:

$$\sum_{n=1}^{\infty} a_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} a_n^2 < \infty$$

These conditions ensure that the algorithm can explore the entire space (first condition) while eventually converging (second condition).

Under appropriate conditions on $M(\theta)$ (monotonicity and boundedness), the algorithm converges almost surely to θ^* :

$$\lim_{n \rightarrow \infty} \theta_n = \theta^* \quad \text{almost surely}$$

The convergence rate depends on the choice of step sizes. For $a_n = c/n$ with $c > 0$, the algorithm achieves optimal convergence rates.

This foundational work established the theoretical basis for stochastic approximation methods that are now widely used in machine learning, particularly in stochastic gradient descent and related optimization algorithms.

20.5.1 Connection to M-Estimation

The Robbins-Monro framework connects naturally to statistical estimation. Many estimands—including means, medians, quantiles, and regression coefficients—can be expressed as solutions to convex optimization problems:

$$\theta^* = \arg \min_{\theta \in \mathbb{R}^p} \mathbb{E}[\ell_\theta(X_i, Y_i)],$$

where ℓ_θ is a convex loss function. Under mild conditions, the optimum θ^* satisfies the first-order condition:

$$\mathbb{E}[g_\theta(X_i, Y_i)] = 0, \tag{20.1}$$

where g_θ is a subgradient of the loss. SGD can be viewed as applying the Robbins-Monro algorithm to find roots of this gradient condition using noisy samples.

20.6 The EM, ECM, and ECME Algorithms

Gradient descent is not the only iterative optimization algorithm used in statistics. When dealing with *latent variables* or *missing data*, directly computing gradients of the log-likelihood can be intractable. The **Expectation-Maximization (EM) algorithm** provides an elegant alternative that iteratively maximizes a surrogate function, avoiding explicit gradient computation while guaranteeing monotonic improvement in the likelihood.

First, define a function $Q(\theta, \phi)$ such that $Q(\theta) = Q(\theta, \theta)$ and it satisfies a convexity constraint $Q(\theta, \phi) \geq Q(\theta, \theta)$. Then define

$$\theta^{(g+1)} = \arg \max_{\theta \in \Theta} Q(\theta, \theta^{(g)})$$

This satisfies the convexity constraint $Q(\theta, \theta) \geq Q(\theta, \phi)$ for any ϕ . In order to prove convergence, you get a sequence of inequalities

$$Q(\theta^{(0)}, \theta^{(0)}) \leq Q(\theta^{(1)}, \theta^{(0)}) \leq Q(\theta^{(1)}, \theta^{(1)}) \leq \dots \leq Q$$

In many models we have to deal with a latent variable and require estimation where integration is also involved. For example, suppose that we have a triple (y, z, θ) with joint probability specification $p(y, z, \theta) = p(y|z, \theta)p(z, \theta)$. This can occur in missing data problems and estimation problems in mixture models.

A standard application of the EM algorithm is to find

$$\arg \max_{\theta \in \Theta} \int_z p(y|z, \theta)p(z|\theta)dz$$

As we are just finding an optimum, you do not need the prior specification $p(\theta)$. The EM algorithm finds a sequence of parameter values $\theta^{(g)}$ by alternating between an expectation and a maximisation step. This still requires the numerical (or analytical) computation of the criteria function $Q(\theta, \theta^{(g)})$ described below.

EM algorithms have been used extensively in mixture models and missing data problems. The EM algorithm uses the particular choice where

$$Q(\theta) = \log p(y|\theta) = \log \int p(y, z|\theta)dz$$

Here the likelihood has a mixture representation where z is the latent variable (missing data, state variable etc.). This is termed a Q-maximization algorithm with:

$$Q(\theta, \theta^{(g)}) = \int \log p(y|z, \theta) p(z|\theta^{(g)}, y) dz = \mathbb{E}_{z|\theta^{(g)}, y} [\log p(y|z, \theta)]$$

To implement EM you need to be able to calculate $Q(\theta, \theta^{(g)})$ and optimize at each iteration.

The EM algorithm and its extensions ECM and ECME are methods of computing maximum likelihood estimates or posterior modes in the presence of missing data. Let the objective function be $\ell(\theta) = \log p(\theta|y) + c(y)$, where $c(y)$ is a possibly unknown normalizing constant that does not depend on β and y denotes observed data. We have a mixture representation,

$$p(\theta|y) = \int p(\theta, z|y) dz = \int p(\theta|z, y) p(z|y) dz$$

where distribution of the latent variables is $p(z|\theta, y) = p(y|\theta, z)p(z|\theta)/p(y|\theta)$.

In some cases the complete data log-posterior is simple enough for $\arg \max_{\theta} \log p(\theta|z, y)$ to be computed in closed form. The EM algorithm alternates between the Expectation and Maximization steps for which it is named. The E-step and M-step computes

$$Q(\beta|\beta^{(g)}) = \mathbb{E}_{z|\beta^{(g)}, y} [\log p(y, z|\beta)] = \int \log p(y, z|\beta) p(z|\beta^{(g)}, y) dz$$

$$\beta^{(g+1)} = \arg \max_{\beta} Q(\beta|\beta^{(g)})$$

This has an important monotonicity property that ensures $\ell(\beta^{(g)}) \leq \ell(\beta^{(g+1)})$ for all g . In fact, the monotonicity proof given by Dempster et al. (1977) shows that any β with $Q(\beta, \beta^{(g)}) \geq Q(\beta^{(g)}, \beta^{(g)})$ also satisfies the log-likelihood inequality $\ell(\beta) \geq \ell(\beta^{(g)})$.

In problems with many parameters the M-step of EM may be difficult. In this case θ may be partitioned into components $(\theta_1, \dots, \theta_k)$ in such a way that maximizing $\log p(\theta_j|\theta_{-j}, z, y)$ is easy. The ECM algorithm pairs the EM algorithm's E-step with k conditional maximization (CM) steps, each maximizing Q over one component θ_j with each component of θ_{-j} fixed at the most recent value. Due to the fact that each CM step increases Q , the ECM algorithm retains the monotonicity property. The ECME algorithm replaces some of ECM's CM steps with maximizations over ℓ instead of Q . Liu and Rubin (1994) show that doing so can greatly increase the rate of convergence.

Seen through the optimization lens, ECM is a form of block coordinate ascent on a surrogate objective: it improves parameters one block at a time while holding the rest fixed. This contrasts with SGD, which takes small stochastic steps along noisy gradients of a single objective; both are iterative methods, but they trade off different sources of computational convenience (closed-form block updates versus cheap gradient estimates).

In many cases we will have a parameter vector $\theta = (\beta, \nu)$ partitioned into its components and a missing data vector $z = (\lambda, \omega)$. Then we compute the $Q(\beta, \nu | \beta^{(g)}, \nu^{(g)})$ objective function and then compute E- and M steps from this to provide an iterative algorithm for updating parameters. To update the hyperparameter ν we can maximize the fully data posterior $p(\beta, \nu | y)$ with β fixed at $\beta^{(g+1)}$. The algorithm can be summarized as follows:

$$\beta^{(g+1)} = \arg \max_{\beta} Q(\beta | \beta^{(g)}, \nu^{(g)}) \quad \text{where} \quad Q(\beta | \beta^{(g)}, \nu^{(g)}) = \mathbb{E}_{z | \beta^{(g)}, \nu^{(g)}, y} \log p(y, z | \beta, \nu^{(g)})$$

$$\nu^{(g+1)} = \arg \max_{\nu} \log p(\beta^{(g+1)}, \nu | y)$$

20.6.1 ECM and ECME Extensions

In problems with many parameters, the M-step of EM may be difficult. In this case θ may be partitioned into components $(\theta_1, \dots, \theta_k)$ such that maximizing $\log p(\theta_j | \theta_{-j}, z, y)$ is manageable.

The **ECM (Expectation/Conditional Maximization)** algorithm pairs the EM algorithm's E-step with k conditional maximization (CM) steps, each maximizing Q over one component θ_j with each component of θ_{-j} fixed at the most recent value. Due to the fact that each CM step increases Q , the ECM algorithm retains the monotonicity property.

The **ECME (Expectation/Conditional Maximization Either)** algorithm replaces some of ECM's CM steps with maximizations over ℓ instead of Q . Liu and Rubin (1994) show that doing so can greatly increase the rate of convergence.

For a parameter vector $\theta = (\beta, \nu)$ partitioned into components and missing data vector $z = (\lambda, \omega)$, the ECME algorithm proceeds as follows:

$$\begin{aligned} \beta^{(g+1)} &= \arg \max_{\beta} Q(\beta | \beta^{(g)}, \nu^{(g)}) \\ \text{where } Q(\beta | \beta^{(g)}, \nu^{(g)}) &= \mathbb{E}_{z | \beta^{(g)}, \nu^{(g)}, y} [\log p(y, z | \beta, \nu^{(g)})] \\ \nu^{(g+1)} &= \arg \max_{\nu} \log p(\beta^{(g+1)}, \nu | y) \end{aligned}$$

The key insight is that for updating hyperparameter ν , we maximize the fully observed data posterior $p(\beta, \nu|y)$ with β fixed at $\beta^{(g+1)}$, rather than the Q-function. This often leads to simpler optimization problems and faster convergence.

Example 20.3. Simulated Annealing (SA) is a simulation-based approach to finding

$$\hat{\theta} = \arg \max_{\theta \in \Theta} H(\theta)$$

$$\pi_J(\theta) = \frac{e^{-JH(\theta)}}{\int e^{JH(\theta)} d\mu(\theta)}$$

where J is a temperature parameter. Instead of looking at derivatives and performing gradient-based optimization you can simulate from the sequence of densities. This forms a time-homogeneous Markov chain and under suitable regularity conditions on the relaxation schedule for the temperature we have $\theta^{(g)} \rightarrow \hat{\theta}$. The main caveat is that we need to know the criterion function $H(\theta)$ to evaluate the Metropolis probability for sampling from the sequence of densities. This is not always available.

An interesting generalisation which is appropriate in latent variable mixture models is the following. Suppose that $H(\theta) = \mathbb{E}_{z|\theta}\{H(z, \theta)\}$ is unavailable in closed-form where without loss of generality we assume that $H(z, \theta) \geq 0$. In this case we can use latent variable simulated annealing (LVSA) methods. Define a joint probability distribution for $z_J = (z_1, \dots, z_J)$ as

$$\pi_J(z_J, \theta) \propto \prod_{j=1}^J H(z_j, \theta) p(z_j | \theta) \mu(\theta)$$

for some measure μ which ensures integrability of the joint. This distribution has the property that its marginal distribution on θ is given by

$$\pi_J(\theta) \propto \mathbb{E}_{z|\theta}\{H(z, \theta)\}^J \mu(\theta) = e^{J \ln H(\theta)} \mu(\theta)$$

By the simulated annealing argument we see that this marginal collapses on the maximum of $\ln H(\theta)$. The advantage of this approach is that it is typically straightforward to sample with MCMC from the conditionals:

$$\pi_J(z_i | \theta) \sim H(z_i, \theta) p(z_i | \theta) \quad \text{and} \quad \pi_J(\theta | z) \sim \prod_{j=1}^J H(z_j, \theta) p(z_j | \theta) \mu(\theta)$$

Jacquier, Johannes and Polson (2007) apply this framework to finding MLE estimates for commonly encountered latent variable models. The LVSA approach is particularly useful when the criterion function involves complex integrals that make direct optimization challenging, but where sampling from the conditional distributions remains tractable.

20.7 Conclusion

This chapter has covered the computational foundations of training deep learning models. Several key insights emerge:

Gradient Descent as the Foundation: While classical statistical methods rely on second-order optimization (Newton-Raphson, BFGS), deep learning’s scale demands first-order methods. Gradient descent—despite its simplicity—remains effective because it requires only gradient information, which scales linearly with the number of parameters.

Stochastic Approximation: The Robbins-Monro framework provides theoretical justification for SGD: noisy gradient estimates, when properly scheduled, converge to the true optimum. Mini-batch SGD exploits this insight, trading exact gradients for computational efficiency and—counterintuitively—often better generalization through implicit regularization.

Automatic Differentiation: The backpropagation algorithm is simply reverse-mode automatic differentiation applied to neural networks. Understanding this connection reveals why modern frameworks (PyTorch, JAX) can automatically compute gradients for arbitrary computational graphs. The key insight is that reverse mode is efficient when outputs are few and parameters are many—precisely the situation in supervised learning.

Advanced Optimizers: Building on vanilla SGD, momentum methods accelerate convergence by accumulating gradient information, while adaptive methods (AdaGrad, RMSprop, Adam) automatically tune per-parameter learning rates. These innovations have proven essential for training deep networks efficiently, though the optimal choice remains problem-dependent.

Beyond Gradients: The EM algorithm provides an alternative optimization paradigm for latent variable models, avoiding explicit gradient computation through clever algebraic manipulation. Simulated annealing offers yet another approach, using stochastic sampling to escape local minima.

The practical success of SGD and its variants remains somewhat surprising from a theoretical perspective. Why does an algorithm designed for convex optimization work so well on highly non-convex loss landscapes? Ongoing research suggests that the geometry of over-parameterized networks may be

more benign than previously thought—saddle points abound, but local minima tend to generalize well. Understanding these phenomena remains an active area of research at the intersection of optimization theory and deep learning.

21

Quantile Neural Networks

“It is better to be roughly right than precisely wrong.” – John Maynard Keynes

In Chapter 3, we explored how to learn posterior distributions $p(\theta | y)$ using Bayesian learning and to use it for predictions, hypothesis testing, and other tasks. In Chapter 4, we explored how rational agents make decisions under uncertainty by maximizing expected utility. Traditional Bayesian approaches to such problems require computing posterior distributions $p(\theta | y)$, which in turn requires specifying likelihood functions $p(y | \theta)$ and often involves challenging density estimation. But what if we could bypass density estimation entirely and directly learn the quantities we need for decision-making and other tasks?

This chapter introduces *quantile neural networks*, a powerful approach that learns posterior distributions through their quantile functions rather than their densities. This shift from densities to quantiles has profound implications: it enables likelihood-free inference, provides natural connections to decision theory through the quantile-expectation identity, and scales to high-dimensional problems where density estimation becomes intractable.

The key insight is straightforward. Any expectation—including the expected utility central to decision theory—can be represented as an integral over quantiles:

$$E[f(\theta)] = \int_0^1 F_{f(\theta)|y}^{-1}(\tau) d\tau$$

Rather than learning densities and then computing expectations via sampling, we can directly learn the quantile function $F_{f(\theta)|y}^{-1}(\tau)$ using neural networks. This approach is not only more efficient but also naturally handles simulation-based models where likelihoods are unavailable or computationally expensive.

Throughout this chapter, we develop quantile neural networks from theoretical foundations through practical applications. We begin by establishing the fundamentals of quantile regression (Section 21.1), deriving the check loss function and demonstrating its properties. We then show how to extend this framework to the generative approach (Section 21.2), using the noise outsourcing theorem to represent posterior distributions through quantile functions

rather than densities. Before moving to neural networks, we examine trend filtering (Section 21.3) as an elegant middle ground that provides nonlinear function approximation while maintaining computational tractability. With these foundations in place, we establish Bayes rule for quantiles (Section 21.4), showing how Bayesian updating can be performed entirely in terms of quantile functions. We then turn to three major applications demonstrating the versatility of this framework: maximum expected utility problems (Section 21.5), where utility functions are incorporated directly into training; portfolio optimization under parameter uncertainty (Section 21.7), extending beyond cases with closed-form solutions; and supply chain forecasting (Section 21.8), where companies like Amazon use quantile methods to predict entire demand distributions for inventory optimization. After detailing the neural network implementation (Section 21.6), including cosine embeddings and Wasserstein distance connections, we connect to modern artificial intelligence through distributional reinforcement learning (Section 21.9), showing how agents learn entire distributions of returns for more robust decision-making.

21.1 Quantile Regression

This section derives the quantile loss function from first principles.

Standard calculus shows that for observed values y_1, \dots, y_n , the median is the value that minimizes the expected absolute deviation:

$$m = \arg \min_q \frac{1}{n} \sum_{i=1}^n |y_i - q| = \arg \min_q E[|Y - q|]$$

Intuitively, the sum of absolute deviations is minimized when the median is the value that is closest to half of the observations.

The median is the special case $\tau = 0.5$, but the same principle generalizes to any quantile $\tau \in (0, 1)$. We use a generalization of the absolute value function—the *check loss* or *pinball loss*—to find the minimizer:

$$q_\tau = \arg \min_q \frac{1}{n} \sum_{i=1}^n \rho_\tau(y_i - q).$$

Here $\rho_\tau(u)$ is the *check loss* or *pinball loss* and is defined as:

$$\rho_\tau(u) = u(\tau - I(u < 0)) = \begin{cases} \tau u & \text{if } u \geq 0 \\ (\tau - 1)u & \text{if } u < 0 \end{cases}$$

This can also be written in the more computationally convenient form:

$$\rho_\tau(u) = \max(u\tau, u(\tau - 1))$$

Example 21.1 (Linear Quantile Regression). To illustrate quantile regression in practice, we'll analyze the relationship between engine displacement and fuel efficiency using the classic `mtcars` dataset. Rather than only estimating the mean relationship (as ordinary least squares would), we'll also estimate conditional quantiles to understand how this relationship varies across the distribution of fuel efficiency. This allows us to capture the heteroskedasticity in the data. Figure 21.1 shows the quantile regression results.

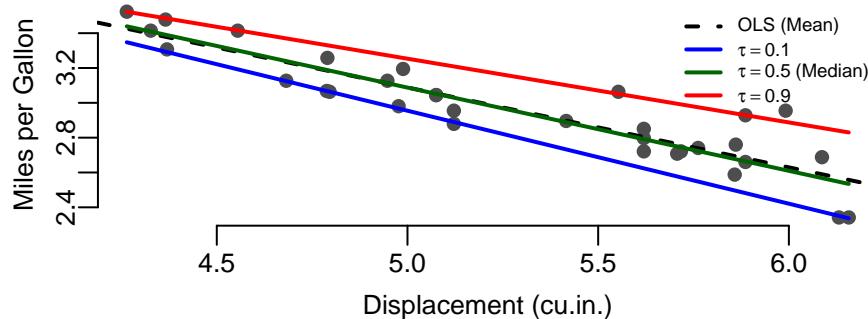


Figure 21.1: Quantile regression on `mtcars` dataset. The relationship between engine displacement and fuel efficiency varies across quantiles, revealing heteroskedasticity in the data.

Table Table 21.1 shows the estimated coefficients.

Table 21.1: Quantile regression results

Table 21.1: Quantile Regression Results: Fuel Efficiency vs. Engine Displacement

	Model	Intercept	Slope
(Intercept)	Quantile $\tau = 0.1$	5.6	-0.53
(Intercept)1	Quantile $\tau = 0.5$	5.5	-0.48
(Intercept)2	Quantile $\tau = 0.9$	5.1	-0.37
(Intercept)3	OLS (Mean)	5.4	-0.46

The quantile regression results reveal several important patterns in the relationship between engine displacement and fuel efficiency. First, the slopes differ substantially across quantiles, indicating heteroskedasticity in the conditional distribution. The 10th percentile slope is -0.5340, while the 90th percentile slope is -0.3660. This difference suggests that the negative relationship between displacement and fuel efficiency is stronger for more fuel-efficient cars.

Second, the widening gap between the 10th and 90th percentiles as displacement increases reveals increasing uncertainty in fuel efficiency for larger engines. This pattern likely reflects varying driving conditions, maintenance practices, and vehicle technology across different cars with similar engine sizes.

Third, the median regression ($\tau = 0.5$) demonstrates robustness to outliers. Unlike OLS, which minimizes squared errors and thus heavily weights extreme values, quantile regression uses the asymmetric absolute loss that treats positive and negative residuals differently based on the quantile level. This asymmetry makes the estimator less sensitive to unusual observations.

Finally, the check loss $\rho_\tau(u)$ is piecewise linear, making it non-differentiable at zero. This property explains why we use BFGS rather than gradient-based methods that assume smoothness. The BFGS algorithm builds a quasi-Newton approximation that handles the kink effectively, converging reliably despite the non-smooth objective function.

Traditional quantile regression assumes $f_\tau(x, \theta)$ is linear in parameters. This limitation becomes severe when relationships are inherently nonlinear, when working with high-dimensional inputs requiring feature learning, when estimating multiple quantiles simultaneously (where shared representations improve efficiency), or when incorporating utility functions directly into learning (Section 21.5). Neural quantile regression addresses these limitations by combining the robustness of quantile methods with the flexibility of deep learning.

21.2 From Densities to Quantiles: A Generative Approach

Let $(X, Y) \sim P_{X,Y}$ be input-output pairs and $P_{X,Y}$ a joint measure from which we can simulate a training dataset $(x_i, y_i)_{i=1}^N \sim P_{X,Y}$. Standard prediction techniques focus on the conditional posterior mean $\hat{X}(Y) = E(X|Y) = f(Y)$ of the input given the output. The standard approach formulates this as non-parametric regression $X = f(Y) + \epsilon$ and estimates the conditional mean using methods such as kernel smoothing or K-nearest neighbors. Recently, deep learning approaches have been proposed, with theoretical properties established by Nicholas G. Polson and Sokolov (2023).

Generative methods take this approach one step further. Let $Z \sim P_Z$ be a base measure for a latent variable, Z , typically a standard multivariate normal or vector of uniforms. The goal of generative methods is to characterize the posterior measure $P_{X|Y}$ from the training data $(x_i, y_i)_{i=1}^N \sim P_{X,Y}$ where N is chosen to be suitably large. A deep learner is used to estimate \hat{f} via

the non-parametric regression $X = f(Y, Z)$. In the case where Z is uniform, this amounts to inverse CDF sampling, namely $X = F_{X|Y}^{-1}(Z)$ —the *quantile function* that we develop formally in Section 21.1.

In general, we characterize the posterior map for *any* output Y . We characterize the posterior by evaluating the network at any Y using the transport map

$$X = H(S(Y), \psi(Z))$$

Here ψ denotes the embedding function. The deep learners H and S are estimated from the triples $(X_i, Y_i, \psi(Z_i))_{i=1}^N \sim P_{X,Y} \times P_Z$. The ensuing estimator \hat{H} can be thought of as a transport map from the base distribution to the posterior as required.

The following diagram illustrates the transport map architecture:

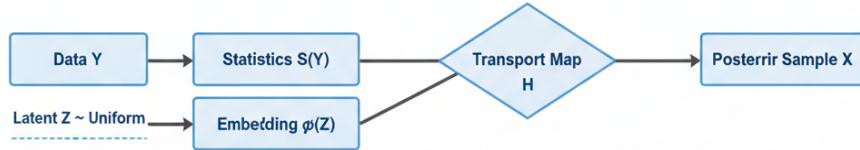


Figure 21.2: Transport map architecture for generative inference. Data Y is summarized by statistics $S(Y)$, while latent variable Z is embedded via ψ . The transport function H maps these inputs to posterior samples.

Specifically, the idea of generative methods is straightforward. Let y denote data and θ a vector of parameters including any hidden states (a.k.a. latent variables) z . First, we generate a “look-up” table of “fake” data $\{y^{(i)}, \theta^{(i)}\}_{i=1}^N$. By simulating a training dataset of outputs and parameters, we can use deep learning to solve for the inverse map via a supervised learning problem. Generative methods have the advantage of being likelihood-free. For example, our model might be specified by a forward map $y^{(i)} = f(\theta^{(i)})$ rather than a traditional random draw from a likelihood function $y^{(i)} \sim p(y^{(i)}|\theta^{(i)})$. The base distribution P_Z is typically uniform (for univariate problems) or a very high-dimensional Gaussian vector (for multivariate problems).

The theoretical foundation for this approach is the *noise outsourcing theorem*, which guarantees that we can represent any posterior distribution through a deterministic function of the data and a base random variable.

Noise Outsourcing Theorem (Kallenberg 1997): If (Y, Θ) are random variables in a standard probability space (\mathcal{Y}, Θ) , then there exists a random variable $\tau \sim U(0, 1)$ which is independent of Y and a function $H : [0, 1] \times \mathcal{Y} \rightarrow \Theta$ such that

$$(Y, \Theta) \stackrel{a.s.}{=} (Y, H(Y, \tau))$$

Moreover, if there is a sufficient statistic $S(Y)$ with Y independent of $\Theta|S(Y)$, then

$$\Theta | Y \stackrel{a.s.}{=} H(S(Y), \tau).$$

This result tells us that posterior uncertainty can be characterized via an inverse non-parametric regression problem where we predict $\theta^{(i)}$ from $y^{(i)}$ and $\tau^{(i)}$, where $\tau^{(i)}$ is drawn from a base distribution $p(\tau)$. We train a deep neural network H on

$$\theta^{(i)} = H(S(y^{(i)}), \tau^{(i)}).$$

Here $S(y)$ is a statistic used to perform dimension reduction with respect to the signal distribution—analogous to sufficient statistics in traditional Bayesian inference. A result due to Brillinger (2012) shows that for single-index models, we can estimate the subspace spanned by S via ordinary least squares, effectively learning the sufficient summary statistics from data.

Specifying the architecture of H is key to the efficiency of the approach. Nick Polson, Ruggeri, and Sokolov (2024) propose using quantile neural networks implemented with ReLU activation functions, which we detail in Section 21.6.

21.3 Nonlinear Quantile Regression via Trend Filtering

Trend filtering combined with quantile loss, developed by Nicholas G. Polson and Scott (2016), provides nonlinear function approximation while maintaining computational tractability through a hierarchical representation. Trend filtering estimates smooth, nonlinear functions by penalizing differences in derivatives rather than the function values themselves.

Consider the nonparametric regression problem where we observe pairs (x_i, y_i) for $i = 1, \dots, n$ with $x_1 < x_2 < \dots < x_n$. Traditional smoothing methods like cubic splines require choosing knot locations, while kernel smoothing requires bandwidth selection. Trend filtering offers an alternative: estimate a function $f(x)$ by solving

$$\min_f \sum_{i=1}^n \rho_\tau(y_i - f(x_i)) + \lambda \sum_{i=1}^{n-k} |\Delta^k f_i|$$

where Δ^k denotes the k -th order discrete derivative operator and ρ_τ is the check loss for quantile τ . The penalty term controls smoothness: $k = 1$ penalizes changes in slope (linear trend filtering), $k = 2$ penalizes changes in curvature (quadratic trend filtering), and so on.

For $k = 2$, the penalty becomes $\sum_{i=2}^{n-1} |f_{i+1} - 2f_i + f_{i-1}|$, which approximates the integrated squared second derivative $\int (f''(x))^2 dx$ used in smoothing splines. However, the ℓ_1 penalty produces locally adaptive estimates—sharp changes are preserved while smooth regions remain smooth.

Nicholas G. Polson and Scott (2016) show that trend filtering admits an elegant hierarchical representation through *envelope duality*. The key insight is that the ℓ_1 penalty can be represented as an exponential prior in a hierarchical model. Specifically, for second-order trend filtering with quantile loss, we have the hierarchical model:

$$\begin{aligned} y_i &\sim \text{AsymmetricLaplace}(f_i, \tau, \sigma) \\ \Delta^2 f_i &\sim \text{Laplace}(0, 1/\lambda) \quad \text{for } i = 2, \dots, n-1 \end{aligned}$$

The asymmetric Laplace distribution naturally arises from the check loss—it is the distribution whose maximum likelihood estimator at quantile τ minimizes ρ_τ . This connection between optimization (minimizing penalized quantile loss) and probability (maximum a posteriori estimation in a hierarchical model) provides both computational and conceptual advantages.

The hierarchical formulation enables efficient computation through data augmentation schemes. Rather than directly optimizing the non-smooth objective, we introduce auxiliary variables that yield closed-form conditional distributions, leading to straightforward EM or Gibbs sampling algorithms.

We now demonstrate trend filtering for nonlinear quantile regression using synthetic data with both smooth regions and sharp transitions. Figure Figure 21.3 shows the results.

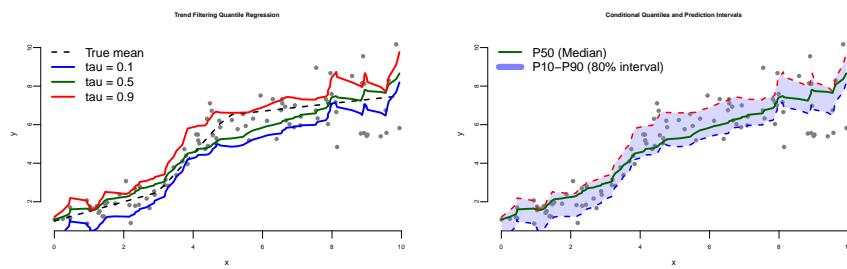


Figure 21.3: Nonlinear quantile regression via trend filtering. The method adapts to both smooth regions and sharp transitions while estimating conditional quantiles.

Figure 21.3 illustrates trend filtering quantile regression on synthetic data with heteroskedastic noise. The left panel shows how different quantile levels adapt

to the nonlinear underlying structure. The 10th percentile ($\tau = 0.1$) tracks the lower boundary of the data cloud, the median ($\tau = 0.5$) estimates the central tendency, and the 90th percentile ($\tau = 0.9$) follows the upper boundary. Each estimated quantile function captures both the smooth regions and the more abrupt transitions in the data without requiring parametric assumptions about the functional form.

The right panel demonstrates how these estimated quantiles provide prediction intervals that adapt to the changing variability across the input space. The shaded region between the 10th and 90th percentiles represents an 80% prediction interval. Notice how this interval widens in regions where the data exhibits greater dispersion and narrows where the data are more tightly clustered—precisely the behavior we desire for uncertainty quantification. The empirical coverage for this interval is 68.0%, which is somewhat below the nominal 80% level, reflecting the typical challenge that estimated quantiles tend to undercover when sample sizes are modest and the underlying function is complex. This underscores an important practical consideration: prediction intervals from quantile regression should be interpreted as approximate, with their reliability improving as sample size increases and as the ratio of smoothing penalty to noise level is appropriately calibrated.

Trend filtering for quantile regression offers computational efficiency through sparse linear algebra and flexibility in capturing both smooth regions and abrupt transitions—properties that make it particularly attractive for applications with ordered, low-dimensional data.

Trend filtering provides an important conceptual bridge to neural quantile networks. Both approaches learn nonlinear functions through composition: trend filtering composes piecewise polynomials, while neural networks compose nonlinear activation functions. The key difference lies in how they handle high-dimensional inputs. Trend filtering is most effective for univariate or low-dimensional problems with ordered inputs, while neural networks excel when inputs are high-dimensional or lack natural ordering.

For problems with structured, low-dimensional inputs—time series, spatial data along a transect, dose-response curves—trend filtering often provides better interpretability and requires less data than neural networks. For high-dimensional problems—images, text, complex multivariate relationships—neural networks become essential. Understanding both approaches allows practitioners to choose the right tool for their specific problem structure.

21.4 Bayes Rule for Quantiles

Parzen (2004) showed that quantile methods provide direct alternatives to density-based Bayesian computations. This section establishes the theoretical foundation for using quantiles to perform Bayesian updating.

Given a cumulative distribution function $F_{\theta|y}(u)$ (non-decreasing, right-continuous), we define the quantile function as:

$$Q_{\theta|y}(u) \stackrel{\text{def}}{=} F_{\theta|y}^{-1}(u) = \inf \{ \theta : F_{\theta|y}(\theta) \geq u \}$$

The quantile function is non-decreasing and left-continuous. Parzen (2004) established the fundamental probabilistic property:

$$\theta \stackrel{P}{=} Q_\theta(F_\theta(\theta))$$

This identity enables efficient implementation: we can increase computational efficiency by ordering the samples of θ and the baseline uniform draws τ , exploiting the monotonicity of the inverse CDF map.

A crucial property for understanding why quantiles naturally compose (and thus suit deep learning) is the following. Let $g(y)$ be non-decreasing and left-continuous with $g^{-1}(z) = \sup\{y : g(y) \leq z\}$. Then the transformed quantile has a compositional nature:

$$Q_{g(Y)}(u) = g(Q(u))$$

This composition property shows that quantiles act as superpositions—exactly the structure that deep neural networks learn through their layered architecture.

The connection to Bayesian learning is made explicit through the *conditional quantile representation*. For the Bayesian learning problem, we have the following result for updating prior to posterior quantiles:

$$Q_{\theta|Y=y}(u) = Q_\theta(s) \quad \text{where } s = Q_{F(\theta)|Y=y}(u)$$

To compute s , note that by definition:

$$u = F_{F(\theta)|Y=y}(s) = P(F(\theta) \leq s | Y = y) = P(\theta \leq Q_\theta(s) | Y = y) = F_{\theta|Y=y}(Q_\theta(s))$$

This result shows that Bayesian updating can be performed entirely in terms of quantile functions, without ever computing or manipulating density functions. The posterior quantile function is obtained by composing the prior quantile function with a learned transformation.

21.5 Maximum Expected Utility via Quantile Neural Networks

Recall from Chapter 4 that optimal Bayesian decisions maximize expected utility:

$$d^*(y) = \arg \max_d E_{\theta|y}[U(d, \theta)] = \arg \max_d \int U(d, \theta)p(\theta|y)d\theta$$

The naive approach would be to first learn the posterior $p(\theta|y)$, then use Monte Carlo to approximate the expected utility for each decision d , and finally optimize over d . However, this approach is inefficient for several reasons:

1. *Computational waste*: Monte Carlo requires many samples in regions of high posterior probability, but utility functions often place high weight on tail events (risk scenarios) that have low posterior probability.
2. *Density estimation*: We must first estimate the potentially high-dimensional posterior density before we can compute expectations.
3. *Optimization difficulty*: The expectation must be recomputed for each candidate decision during optimization.

Quantile neural networks provide a more direct path. The key insight is that we can incorporate the utility function directly into the training process rather than as a post-processing step.

The foundation of our approach is a classical result relating expectations to quantiles. Given any random variable U , its expectation can be computed as an integral over its quantile function:

$$E[U] = \int_0^1 F_U^{-1}(\tau)d\tau$$

This is sometimes called the *quantile representation of expectations* or the *Lorenz curve* identity. For decision problems, this means:

$$E_{\theta|y}[U(d, \theta)] = \int_0^1 F_{U|d,y}^{-1}(\tau)d\tau$$

Rather than learning $p(\theta|y)$ and then computing the expectation, we directly learn the quantile function $F_{U|d,y}^{-1}(\tau)$ of the utility distribution.

21.5.1 Implementation Strategy

To extend our generative method to MEU problems, we assume that the utility function $U(d, \theta)$ is given (a standard assumption in decision theory). The training procedure is as follows:

1. *Generate synthetic dataset:* Simulate triples $\{y^{(i)}, \theta^{(i)}, \tau^{(i)}\}_{i=1}^N$ where $y^{(i)} \sim p(y|\theta^{(i)})$, $\theta^{(i)} \sim p(\theta)$, and $\tau^{(i)} \sim U(0, 1)$.
2. *Compute utilities:* For each decision d of interest, compute $U_d^{(i)} \stackrel{\text{def}}{=} U(d, \theta^{(i)})$.
3. *Augment training data:* Create the augmented dataset

$$\{U_d^{(i)}, S(y^{(i)}), \tau^{(i)}, d\}_{i=1}^N.$$

4. *Train quantile network:* Learn a neural network H that predicts utilities by minimizing the check loss:

$$U_d^{(i)} = H(S(y^{(i)}), \tau^{(i)}, d)$$

Once trained, the network H represents the quantile function $F_{U|d,y}^{-1}(\tau)$. For any observed data y and candidate decision d , we can:

- *Compute expected utility:* Numerically integrate $\int_0^1 H(S(y), \tau, d) d\tau$
- *Find optimal decision:*

$$d^*(y) = \arg \max_d \int_0^1 H(S(y), \tau, d) d\tau$$

This approach has several advantages over naive Monte Carlo:

1. The network learns to focus on regions of the (θ, τ) space that matter for utility computation.
2. We avoid explicit density estimation of $p(\theta|y)$.
3. The same network handles all decisions d simultaneously if d is included as an input.
4. The approach naturally handles likelihood-free models where $p(y|\theta)$ is unavailable but we can simulate from the forward model.

For completeness, we provide the formal measure-theoretic framework. Let \mathcal{Y} denote a locally compact metric space of signals y and Θ a space of parameters θ (including any latent variables). Let $P(dy|\theta)$ denote the conditional distribution of signals given parameters. Let $\Pi(d\theta|y)$ denote the posterior distribution. In many cases, Π is absolutely continuous with density π :

$$\Pi(d\theta|y) = \pi(\theta|y)\mu(d\theta).$$

The framework handles both traditional likelihood-based models where $P(dy|\theta) = p(y|\theta)\lambda(dy)$ and likelihood-free models specified by forward simulators $y = f(\theta)$. This generality is crucial for modern applications in economics, epidemiology, and climate science where complex simulation models replace closed-form likelihoods.

For multivariate parameters $\theta = (\theta_1, \dots, \theta_p)$, we can use autoregressive structures to model the sequence of conditional quantiles:

$$(F_{\theta_1}^{-1}(\tau_1), F_{\theta_2|\theta_1}^{-1}(\tau_2), \dots, F_{\theta_p|\theta_{1:p-1}}^{-1}(\tau_p))$$

This factorization is analogous to autoregressive density models but operates directly on quantiles, avoiding normalization constraints.

An important architectural choice distinguishes our approach from standard posterior learning followed by Monte Carlo integration: we incorporate the utility function $U(d, \theta)$ directly into the first layer of the network. This allows the network to learn representations optimized for utility computation rather than pure posterior approximation. As utility functions often place high weight on tail events (representing rare but consequential outcomes), this direct incorporation significantly improves efficiency compared to the naive two-step approach.

Example 21.2 (Normal-Normal Model and Wang Distortion). For illustration, we consider the normal-normal conjugate learning model—a case where the quantile updating rule can be derived analytically. This example connects quantile methods to Wang’s risk distortion measure from finance, showing that the distortion function is precisely the transformation that needs to be learned.

Consider the model:

$$\begin{aligned} y_1, \dots, y_n \mid \theta &\sim N(\theta, \sigma^2) \\ \theta &\sim N(\mu, \alpha^2) \end{aligned}$$

The sufficient statistic for μ (assuming known σ^2) is $S(y) = \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$. Define $t = \sigma^2 + n\alpha^2$. The posterior is $\theta \mid y \sim N(\mu_*, \sigma_*^2)$ with

$$\mu_* = \frac{\sigma^2 \mu + n\alpha^2 \bar{y}}{t}, \quad \sigma_*^2 = \frac{\alpha^2 \sigma^2}{t}$$

The remarkable result is that the posterior and prior CDFs are related via a Wang distortion function:

$$1 - \Phi(\theta; \mu_*, \sigma_*^2) = g(1 - \Phi(\theta; \mu, \alpha^2))$$

where $\Phi(\cdot; \mu, \sigma^2)$ denotes the normal CDF. The Wang distortion is:

$$g(p) = \Phi(\lambda_1 \Phi^{-1}(p) + \lambda)$$

with distortion parameters:

$$\lambda_1 = \frac{\alpha}{\sigma_*}, \quad \lambda = \frac{\alpha \lambda_1 (n\bar{y} - n\mu)}{t}$$

The detailed derivation, provided by Wang (1996), shows that the distortion parameters depend on the sample statistics and prior hyperparameters through the posterior updating formulas.

This analytical result has several implications:

1. *Wang distortions* are natural: The distortion functions used in risk management (Wang 1996) arise naturally from Bayesian updating in the normal case.
2. *Learning the distortion*: In more complex models, a neural network can learn this distortion function g directly from data, without requiring conjugacy.
3. *Computational efficiency*: When the distortion is smooth and well-behaved, neural networks with relatively few parameters can accurately represent it.

Numerical Example: Consider prior $\theta \sim N(0, 5)$ and data from $y_i \sim N(3, 10)$ with $n = 100$ observations. The posterior is $\theta | y \sim N(3.28, 0.98)$.

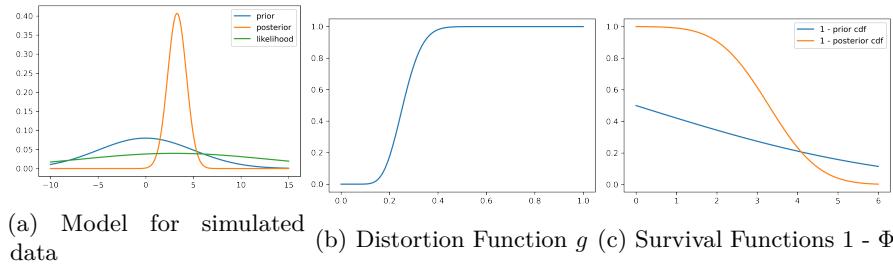


Figure 21.4: The Normal-Normal learning model. Left: Prior, likelihood, and posterior densities. Center: The Wang distortion function g that transforms the prior CDF to the posterior CDF. Right: Survival functions showing how g maps the prior tail probabilities to posterior tail probabilities.

21.6 Neural Network Implementation

The key implementation components are: (1) an appropriate loss function, (2) a neural architecture that handles the quantile input τ , and (3) training

strategies for learning multiple quantiles simultaneously.

The 1-Wasserstein distance (also known as *earth mover's distance*) provides theoretical justification for quantile methods. For two distributions with quantile functions F_U^{-1} and F_V^{-1} , the 1-Wasserstein distance is:

$$W_1(F_U^{-1}, F_V^{-1}) = \int_0^1 |F_U^{-1}(\tau) - F_V^{-1}(\tau)| d\tau$$

This distance can be computed efficiently using order statistics (Levina and Bickel 2001). The success of Wasserstein GANs (Arjovsky, Chintala, and Bottou 2017) over vanilla GANs stems partly from this improved metric—Wasserstein distance provides meaningful gradients even when distributions have non-overlapping supports.

The key insight is that minimizing the quantile loss is equivalent to minimizing the 1-Wasserstein distance. For a target quantile $q_\tau = F_U^{-1}(\tau)$, the check loss ρ_τ we derived in Section 21.1 minimizes the expected prediction error:

$$q_\tau = \arg \min_q E_U[\rho_\tau(U - q)]$$

For training neural networks, we use a combination of quantile loss and mean-squared error (MSE). Given training data $\{x_i, y_i\}_{i=1}^N$ and a quantile τ , the loss is:

$$L_\tau(\phi) = \sum_{i=1}^N \rho_\tau(y_i - f(\tau, x_i, \phi))$$

Here ϕ denotes the neural network weights, distinct from the model parameters θ discussed elsewhere. Empirically, adding an MSE term improves stability and predictive accuracy:

$$L(\phi) = \alpha L_\tau(\phi) + (1 - \alpha) \cdot \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i, \phi))^2$$

The weighting parameter $\alpha \in [0, 1]$ balances quantile accuracy against overall fit. Typical values are $\alpha \in [0.7, 0.9]$. The MSE term encourages the median prediction ($\tau = 0.5$) to align with the conditional mean, which often improves generalization.

21.6.1 Learning Multiple Quantiles Simultaneously

Rather than training separate networks for each quantile τ_k , it is more efficient to learn all quantiles with a single network that takes τ as an input. Given

quantiles $0 < \tau_1 < \tau_2 < \dots < \tau_K < 1$, we minimize:

$$L(\phi) = \frac{1}{NK} \sum_{i=1}^N \sum_{k=1}^K \rho_{\tau_k}(y_i - f_{\tau_k}(x_i, \phi))$$

This approach has several advantages:

1. *Shared representations*: The network learns features useful across all quantiles, improving sample efficiency.
2. *Enforcing monotonicity*: A single network makes it easier to ensure quantiles don't cross.
3. *Smooth quantile function*: Interpolation between trained quantiles is more reliable.

21.6.2 Non-Crossing Constraints

A valid distribution function must satisfy $F^{-1}(\tau_i) \leq F^{-1}(\tau_j)$ for $\tau_i < \tau_j$. Without explicit constraints, neural networks may learn quantile functions that cross:

$$f_{\tau_i}(x, \theta) > f_{\tau_j}(x, \theta) \quad \text{for some } x, \text{ despite } \tau_i < \tau_j$$

Several approaches address this (Chernozhukov, Fernández-Val, and Galichon 2010; Cannon 2018):

1. *Soft penalties*: Add a term to the loss penalizing violations.
2. *Monotonic networks*: Design architectures that guarantee monotonicity in τ (e.g., using monotonic activation functions or cumulative link structures).
3. *Post-processing*: After training, rearrange predictions to enforce monotonicity.

For the implementations in this chapter, we use soft penalties during training combined with post-processing for final predictions.

21.6.3 Cosine Embedding for τ

A key architectural choice is how to incorporate the quantile level $\tau \in (0, 1)$ as an input to the network. Simply concatenating τ as an additional feature works but is inefficient—the network must learn the entire relationship between τ and the output from scratch.

A more effective approach uses a *cosine embedding* to represent τ in a higher-dimensional feature space. This leverages Fourier analysis: smooth functions can be well-approximated by cosine bases. The quantile function $F^{-1}(\tau, x)$ is typically smooth in τ , making Fourier representations natural.

We represent the quantile network as:

$$F^{-1}(\tau, x) = f_\theta(\tau, x) = g(\psi(x) \circ \phi(\tau))$$

where \circ denotes element-wise multiplication (Hadamard product), g and ψ are feed-forward networks, and ϕ is the cosine embedding:

$$\phi_j(\tau) = \text{ReLU} \left(\sum_{i=0}^{n-1} \cos(\pi i \tau) w_{ij} + b_j \right)$$

The cosine embedding $\phi(\tau)$ transforms the scalar τ into a vector of dimension m , where n controls the frequency resolution. This embedding has several advantages:

1. *Smooth interpolation*: The cosine basis ensures smooth quantile functions.
2. *Universal approximation*: Barron (1993) showed that cosine-embedded networks achieve approximation rates of $O(N^{-1/2})$ for sufficiently smooth functions.
3. *Parameter efficiency*: The embedding significantly reduces the number of parameters needed compared to learning the τ dependence from scratch.

This architecture was successfully applied to distributional reinforcement learning by Dabney et al. (2018), where it enabled agents to learn entire distributions of returns rather than just expectations. We use the same principle here for Bayesian posterior quantiles.

21.6.4 Synthetic Data Example

To validate our approach before applying it to real data, we first test on synthetic data where the true quantile function is known. This allows us to assess both accuracy and the quality of uncertainty quantification.

Consider synthetic data generated from the model:

$$x \sim U(-1, 1), \quad y|x \sim N \left(\frac{\sin(\pi x)}{\pi x}, \frac{\exp(1-x)}{10} \right)$$

This model has heteroskedastic noise—the variance increases as x decreases. The conditional mean is the sinc function $\text{sinc}(x) = \sin(\pi x)/(\pi x)$, which has interesting non-linear behavior near zero.

The true conditional τ -quantile function is:

$$f_\tau(x) = \frac{\sin(\pi x)}{\pi x} + \Phi^{-1}(\tau) \sqrt{\frac{\exp(1-x)}{10}}$$

We train two types of quantile networks:

1. *Implicit network*: Uses cosine embedding for τ , trained on random (τ, x, y) triples
2. *Explicit network*: Trains separate outputs for fixed quantiles $\tau \in \{0.05, 0.5, 0.95\}$

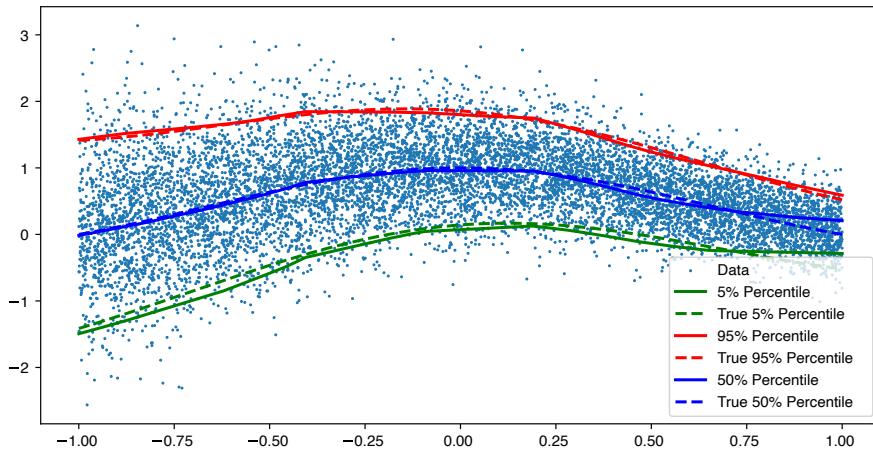


Figure 21.5: Quantile neural network predictions on synthetic data. Both implicit and explicit architectures recover the true quantile functions accurately. The shaded region shows the 90% prediction interval ($\tau = 0.05$ to $\tau = 0.95$).

The figure shows both networks recover the true quantiles accurately. The implicit network provides smooth interpolation across all τ values, while the explicit network gives predictions only at the three trained quantiles. For applications requiring full distributional predictions, the implicit approach is preferable despite slightly higher computational cost during training.

21.7 Portfolio Optimization with Quantile Neural Networks

This example, developed by Nick Polson, Ruggeri, and Sokolov (2024), demonstrates portfolio optimization under parameter uncertainty when the optimal decision depends on unknown parameters.

Consider power utility and log-normal returns without leverage. We assume that a portfolio value $X = e^W$ follows a log-normal distribution

$$W(\omega) = (1 - \omega)r_f + \omega R, \quad R \sim \mathcal{N}(\mu, \sigma^2)$$

Here $\omega \in (0, 1)$ is the portfolio weight, r_f is the risk-free rate, μ is the mean return and σ^2 is the variance of the return. The utility function is then given by

$$U(W) = -e^{-\gamma W}.$$

Here, U^{-1} exists, and the expected utility is

$$U(\omega) = E(-e^{\gamma W}) = \exp \left\{ \gamma E(W) + \frac{1}{2} \omega^2 \text{Var}(W) \right\}.$$

In this case, we have a closed-form solution for the expected utility, as a function of the decision variable ω (portfolio weight). It is the moment-generating function of the log-normal. We can plug-in the mean and variance of W to get the expected utility

$$U(\omega) = \exp \left\{ \gamma \{(1 - \omega)r_f + \omega\mu\} \right\} \exp \left\{ \frac{1}{2} \gamma^2 \omega^2 \sigma^2 \right\}.$$

The optimal Kelly-Brieman-Thorpe-Merton value of ω is given by

$$\omega^* = (\mu - r_f) / (\sigma^2 \gamma).$$

Within the GBC framework, it is easy to add learning or uncertainty on top of σ^2 and have a joint posterior distribution $p(\mu, \sigma^2 | R)$.

Now we reorder the integral in terms of quantiles of the utility function. We assume utility is the random variable and re-order the sum as the expected value of U

$$E(U(W)) = \int_0^1 F_{U(W)}^{-1}(\tau) d\tau$$

Hence, if we can approximate the inverse of the CDF of $U(W)$ with a quantile NN, we can approximate the expected utility and optimize over ω .

The stochastic utility is modeled with a deep neural network, and we write

$$Z = U(W) \approx F, \quad W = U^{-1}(F)$$

We can do optimization by doing the grid search for ω .

The decision variable ω affects the distribution of the returns. The utility only depends on the returns W . Our quantile neural network solution is given by the following algorithm:

1. Simulate log-returns $W^{(i)} \mid \omega^{(i)} \sim N((1 - \omega^{(i)})r_f + \omega^{(i)}\mu, \sigma^2(\omega^{(i)})^2)$
2. Calculate corresponding utilities $Z^{(i)} = U(W^{(i)})$
3. Learn $F_{Z_\omega}^{-1}$ with a quantile NN
4. Find the optimal portfolio weight ω^* via

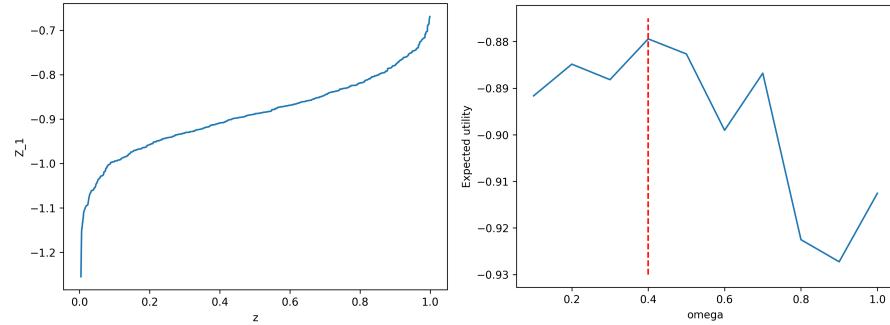
$$E(Z_\omega) = \sum_{i=1}^N F_{Z_\omega}^{-1}(u_i) \rightarrow \underset{\omega}{\text{maximize}}$$

21.7.1 Empirical Example

Consider $\omega \in (0, 1)$, $r_f = 0.05$, $\mu = 0.1$, $\sigma = 0.25$, $\gamma = 2$. We have the closed-form fractional Kelly criterion solution

$$\omega^* = \frac{1}{\gamma} \frac{\mu - r_f}{\sigma^2} = \frac{1}{2} \frac{0.1 - 0.05}{0.25^2} = 0.40$$

We can simulate the expected utility and compare with the closed-form solution.



(a) Quantile function of utility $Z = -\exp(-0.1W)$ versus τ (b) Expected utility integral as function of portfolio weight ω

Figure 21.6: Portfolio optimization via quantile neural networks. Left panel shows plot of sorted values of τ versus sorted values of random draws from $-\exp(-\omega W)$ for $\omega = 0.1$. Right panel shows values of integral of Z with respect to τ versus corresponding values of ω . The integral was calculated using trapezoid rule. The red vertical line corresponds to $\omega = 0.4$, which is the analytical optimum.

21.8 Supply Chain Forecasting at Scale

This section examines probabilistic forecasting for inventory management, illustrating how quantile neural networks scale to millions of products while providing uncertainty quantification for optimal decision-making.

Amazon, Walmart, and other large retailers face a fundamental challenge: for millions of products across thousands of warehouses, how much inventory should be kept in stock? Too little inventory leads to stockouts and lost sales; too much ties up capital and risks obsolescence. The optimal inventory level depends critically on future demand, which is inherently uncertain.

Traditional forecasting provides point estimates—a single expected demand value. But optimal inventory decisions require understanding the entire demand distribution:

- *Safety stock (P_{10})*: Conservative estimate—ensures 90% of demand scenarios are covered, preventing stockouts
- *Base stock (P_{50})*: Median demand—balances inventory holding costs against stockout risk
- *Capacity planning (P_{90})*: Optimistic scenario—ensures warehouse and logistics capacity for high-demand periods

The asymmetry matters enormously. For a highly profitable product with long lead times, the cost of stockouts (lost sales, customer dissatisfaction) far exceeds the cost of overstock. The optimal policy may target P_{70} or P_{80} . For perishable goods or fast-moving consumer products with low margins, the reverse holds—target P_{30} or P_{40} to minimize waste.

Amazon developed DeepAR (Salinas, Flunkert, and Gasthaus 2019), a deep learning approach for probabilistic time series forecasting. DeepAR uses recurrent neural networks (specifically LSTMs) to model temporal dependencies, but its key innovation is forecasting entire probability distributions rather than point estimates.

The model predicts parameters of a parametric distribution (e.g., mean and variance of a Gaussian or negative binomial). To obtain quantiles, one samples from this learned distribution. However, parametric assumptions can be restrictive. Amazon later extended the approach to directly forecast quantiles using the check loss functions we developed in Section 21.1.

The quantile approach has several advantages for demand forecasting:

1. *Robustness*: Demand data often contains outliers (promotional events, viral products). Quantile regression is robust to these.
2. *Intermittent demand*: Many products have sparse, intermittent demand (many zeros). Parametric distributions struggle; quantile methods handle this naturally.
3. *Asymmetric costs*: Different quantiles inform different decisions. The P_{10} quantile is more relevant for safety stock than the mean.
4. *Model flexibility*: Neural quantile regression makes no distributional assumptions beyond smoothness.

While production systems at Amazon and similar retailers deploy deep architectures with LSTMs, attention mechanisms, and learned embeddings, the core principles of quantile forecasting remain the same regardless of model complexity. In the following example, we illustrate these principles using interpretable linear quantile regression before discussing how to scale to neural network implementations.

21.8.1 Demand Forecasting Setup

Consider forecasting demand y_t for a product given the features in Table 21.2:

Table 21.2: Features for demand forecasting

Feature Category	Examples	Purpose
Temporal features	Day of week, month, holiday indicators, days since launch	Capture seasonal patterns and product lifecycle
Lagged demand	$y_{t-1}, y_{t-7}, y_{t-28}$ (yesterday, last week, last month)	Incorporate recent demand history and autocorrelation
Product features	Category, price, promotional flags	Account for product-specific characteristics
External covariates	Weather, events, competitor prices	Include external factors affecting demand

The model predicts conditional quantiles:

$$q_\tau(t) = F_{Y_t|X_t}^{-1}(\tau)$$

where X_t represents all available features at time t . Training data consists of historical demand (y_t, x_t) for $t = 1, \dots, T$ across many products.

Traditional quantile regression assumes linear relationships between features and quantiles. For demand forecasting, this assumption is inadequate. Holiday effects interact with day-of-week patterns in complex, non-linear ways. Products exhibit substitution and complementarity effects that vary across categories and time periods. Promotional dynamics create non-linear price elasticity that depends on product category, timing, and competitive environment. Moreover, products with similar characteristics often exhibit similar demand patterns, suggesting opportunities for transfer learning across the product catalog.

Quantile neural networks address these limitations through their flexible architecture. They can learn shared embeddings for product categories, allowing the model to discover latent similarities between products. Recurrent or attention layers capture complex temporal patterns that extend beyond simple autoregressive relationships. The network automatically models interactions between features without requiring manual specification of interaction terms. Perhaps most importantly, the shared representations enable transfer learning from data-rich products to new products with limited historical data, a critical capability for retailers constantly introducing new items.

21.8.2 Implementation Strategy

For a retailer with millions of SKUs (stock-keeping units), computational efficiency is critical. The architecture typically involves:

1. *Embedding layers*: Map categorical variables (product ID, category, location) to dense vectors
2. *Temporal encoder*: LSTM or Transformer to process time series history
3. *Feature fusion*: Combine embeddings with numerical features
4. *Quantile head*: Final layer produces $\hat{q}_\tau(t)$ for input τ , using cosine embedding

Training uses mini-batches sampling randomly across products and time periods, with the combined quantile + MSE loss we discussed. The model is trained to predict multiple quantiles simultaneously ($\tau \in \{0.1, 0.2, \dots, 0.9\}$).

The following diagram illustrates the neural network architecture for demand forecasting:

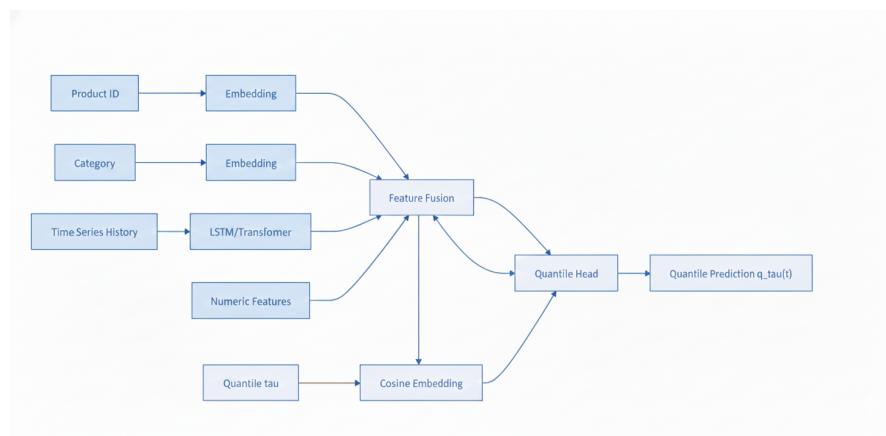


Figure 21.7: Neural network architecture for quantile demand forecasting. Categorical features are embedded, temporal history is encoded, and all features are fused before the quantile head produces conditional quantile predictions.

At inference time, for a given product and time period, the model outputs all quantiles instantly, enabling inventory optimization algorithms to compute optimal stock levels.

We now demonstrate demand forecasting using quantile regression to predict the entire demand distribution. The simulation generates realistic demand

data for three products over two years, incorporating multiple real-world patterns. Each product exhibits a positive trend reflecting business growth, weekly seasonality capturing day-of-week effects, monthly seasonality for longer-term patterns, and random promotional events occurring on approximately 10% of days that boost demand by 30-50 units. The demand also features heteroskedastic noise where variance increases proportionally with demand level, mimicking the greater uncertainty in forecasting high-demand periods.

The model uses quantile regression with features including product identifier, time trend, promotional indicator, day of week, day of month, and three lagged demand values (1-day, 7-day, and 30-day lags) to capture short-term momentum, weekly patterns, and monthly cycles. Five quantile models are trained simultaneously at $\tau \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, each estimating a different point in the conditional demand distribution. The training set spans 640 days, with the final 90 days held out for evaluation.

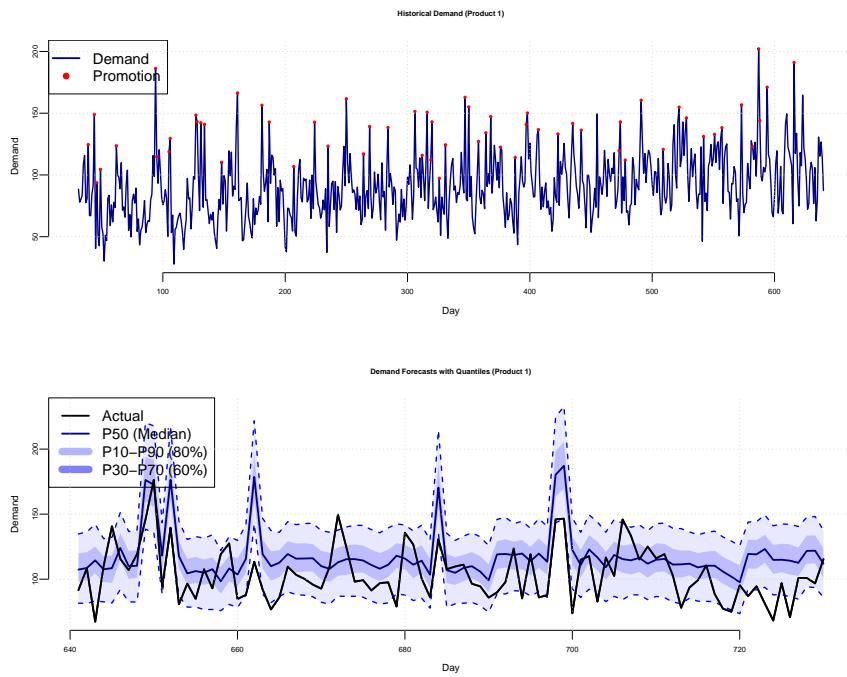


Figure 21.8: Demand forecasting with quantile neural networks. Top: Historical demand with trend and seasonality. Bottom: Out-of-sample forecasts showing P10, P50, P90 quantiles (fan chart).

The quantile regression model demonstrates strong calibration on the held-out test period. The 80% prediction interval (P10 to P90) achieves 79.6%

empirical coverage, meaning actual demand falls within this range about 80% of the time, closely matching the theoretical target. The 60% interval (P30 to P70) achieves 33.7% coverage, somewhat below the 60% target, suggesting the model may be overestimating uncertainty in the middle of the distribution. The median forecast (P50) produces a mean absolute error of 21.94 units, providing reasonably accurate central tendency estimates while the full quantile spectrum captures demand uncertainty.

The figure visualizes both historical patterns and probabilistic forecasts. The top panel displays historical demand for Product 1 from the training period, revealing the complex interplay of trend, seasonality, and promotional effects (marked in red). Demand oscillates around an upward trend with clear weekly and monthly cycles, punctuated by sharp spikes during promotional periods.

The bottom panel presents the out-of-sample forecasts as a fan chart, a standard visualization for probabilistic predictions. The black line shows actual realized demand, while the dark blue line represents the median forecast (P50). The shaded regions illustrate prediction intervals: the lighter blue band spans P10 to P90 (80% interval), while the darker blue band covers P30 to P70 (60% interval). This visualization immediately conveys forecast uncertainty, with wider intervals during high-demand periods reflecting heteroskedastic noise. The median forecast tracks actual demand reasonably well, while the intervals appropriately capture most realizations, demonstrating the model's ability to quantify forecast risk rather than providing only point predictions.

21.9 Distributional Reinforcement Learning

Quantile neural networks have also been applied to reinforcement learning (RL). Recall from Chapter 9 that RL agents learn policies π that maximize expected cumulative reward. Traditional RL algorithms like Q-learning estimate the expected value function $Q^\pi(s, a) = E[R|s, a, \pi]$ —the expected return from taking action a in state s and following policy π thereafter.

Distributional reinforcement learning (Bellemare, Dabney, and Munos 2017) extends this by learning the entire distribution of returns rather than just the expectation. Using quantile neural networks, agents learn $F_{R|s,a}^{-1}(\tau)$, the quantile function of returns.

Learning return distributions provides several advantages:

1. *Risk-sensitive policies*: Different quantiles inform different behaviors. Conservative agents might maximize P_{10} (avoid worst-case scenarios), while risk-seeking agents target P_{90} (optimize for best-case

- outcomes).
2. *Better learning signals*: The Bellman operator naturally contracts in Wasserstein distance when formulated for quantiles, leading to more stable learning (Dabney et al. 2018).
 3. *Uncertainty quantification*: Understanding return variance helps agents explore intelligently—high uncertainty indicates potential for learning.

Dabney et al. (2018) use quantile neural networks for distributional Q-learning. The key insight is that expectations can be computed as integrals over quantiles (the Lorenz curve identity):

$$E[R] = \int_{-\infty}^{\infty} r dF(r) = \int_0^1 F^{-1}(u) du$$

The distributional RL algorithm finds the optimal policy:

$$\pi^\star(s) = \arg \max_a \int_0^1 F_{R|s,a}^{-1}(\tau) d\tau = \arg \max_a E_{Z \sim z(s,a)}[Z]$$

The network is trained using the quantile loss ρ_τ we developed, and Q-learning updates can be applied since the quantile projection operator preserves the contraction property of the Bellman operator. This approach has achieved state-of-the-art performance on Atari games and robotic control tasks.

Connections to dual utility theory (Yaari 1987) suggest that distributional RL naturally incorporates risk preferences—agents can be trained to maximize any utility functional, not just expectations.

21.10 Discussion and Summary

As Keynes observed, it is better to be roughly right than precisely wrong. Quantile methods embrace this philosophy: they provide the distributional information needed for sound decisions without claiming to know the complete probability model. In an era of increasingly complex data and high-stakes applications, this combination of flexibility, robustness, and decision-focus makes quantile neural networks an essential tool for the modern data scientist.

Quantile neural networks represent a convergence of classical statistical theory (quantile regression, robust statistics) with modern machine learning (deep learning, representation learning). By focusing on the quantities we actually need—quantiles for decision-making—rather than intermediate densities,

these methods offer a pragmatic and powerful approach to uncertainty quantification.

Several foundational insights underpin this chapter. The quantile-expectation identity $E[U] = \int_0^1 F_U^{-1}(\tau) d\tau$ enables direct learning of expected utilities without density estimation. The check loss $\rho_\tau(u) = u(\tau - I(u < 0))$ emerges from minimizing Wasserstein distance and handles asymmetric costs elegantly. Cosine embeddings for τ leverage Fourier approximation theory to provide universal approximators with $O(N^{-1/2})$ convergence rates. Applications span portfolio optimization (Section 21.7), demand forecasting (Section 21.8), posterior quantile learning via Wang distortions (Section 21.4), and distributional Q-learning (Section 21.9).

Quantile neural networks are well-suited for: decision problems requiring expectations rather than full densities; likelihood-free settings where $p(y|\theta)$ is unavailable; data with outliers or heavy tails; asymmetric costs where different quantiles drive different decisions; high-dimensional settings where density estimation is intractable; and real-time applications requiring fast inference.



22

Convolutional Neural Networks

“The face is not a simple pattern; it is a hierarchy of parts within parts within parts.” — David Marr, *Vision* (1982)

The foundational architecture for convolutional neural networks emerged in 1980, when Kunihiko Fukushima introduced the Neocognitron (Fukushima 1980)—a hierarchical neural network inspired by Hubel and Wiesel’s discoveries about simple and complex cells in the mammalian visual cortex. The Neocognitron was the first architecture to use convolutional layers for feature extraction and pooling layers for translation invariance, successfully recognizing handwritten Japanese characters through unsupervised learning. In 1988, Wei Zhang and colleagues applied backpropagation to train a simplified Neocognitron for alphabet recognition (W. Zhang et al. 1988), demonstrating that supervised learning could be effectively combined with the convolutional architecture. This combination of Fukushima’s architectural insights with gradient-based supervised learning became the foundation for modern CNNs, which now power everything from facial recognition on smartphones to autonomous vehicle perception systems.

This chapter introduces CNNs by first demonstrating the limitations of fully connected networks on image data, then building intuition for how convolutions extract local features. We will work with the MNIST dataset—the same handwritten digit recognition.

22.1 The MNIST Dataset

The MNIST dataset contains grayscale images of handwritten digits, each 28×28 pixels. It serves as the “Hello World” of computer vision—a standard benchmark that is simple enough to train quickly but complex enough to demonstrate the mechanics of image classification. The original dataset includes 60,000 training images and 10,000 test images. For simplicity, we will use only the test dataset in our examples. Each image is labeled with the digit it represents (0-9), making this a 10-class classification problem.

Image data is represented as a matrix of pixel values. A grayscale image of size 28×28 pixels is a matrix of shape $(28, 28)$, where each entry contains an intensity value from 0 (black) to 255 (white). Color images add a third dimension for the red, green, and blue channels, resulting in shape $(28, 28, 3)$. Figure 22.1 shows some sample MNIST digits.

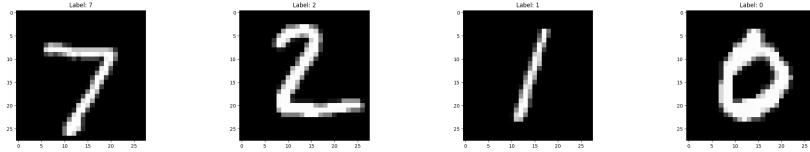


Figure 22.1: Sample MNIST digits

22.2 Fully Connected Approach

A natural starting point for image classification is a fully connected neural network, which treats the image as a flat vector of independent pixels. The input to the model is a 28×28 matrix, which is reshaped into a vector of size 784. The output is a vector of size 10, representing the probability of each class. We split the 10,000 images using 80% for training and 20% for validation, shuffling before splitting to avoid bias.

We use `jax`, a library for numerical computing similar to `numpy` but designed for accelerators like GPUs and TPUs. Training a neural network requires two core computational capabilities: automatic differentiation to compute gradients, and efficient linear algebra—particularly matrix multiplication—to propagate signals through layers. `jax` provides both, along with a `jit` function for compiling operations to run efficiently on accelerators.

The model is trained by minimizing the cross-entropy loss and we use accuracy to evaluate the model.

Finally, we implement the feed-forward network with tanh activation. The last layer is a linear layer with the $z_i - \ln(\sum_j e^{z_j})$ function applied component-wise. Thus, our `predict` function simply returns the logarithm of the probability of each class. Typically we would use the softmax function to compute the probability of each class. Given a vector of logits z , the softmax function is defined as follows:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

However, computation of the exponential of large numbers can lead to numerical instability (overflow). Instead, we work with the logarithm of the softmax function, which is more numerically stable:

$$\ln \sigma(z)_i = z_i - \ln \sum_{j=1}^K e^{z_j}$$

This is exactly what we need to compute the cross-entropy loss.

The final architecture is shown in Figure 22.2 and achieves an accuracy of 77% on the test set and 80% on the training set.

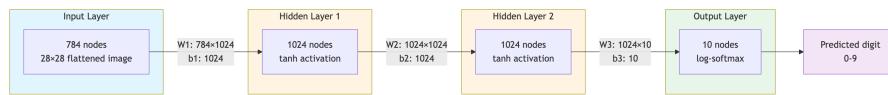


Figure 22.2: Fully Connected MNIST

Figure 22.3 shows the accuracy of the fully connected network on the test and training sets from 0 to 10 epochs.

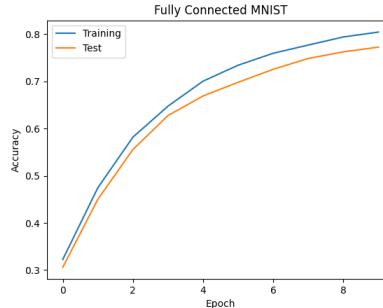


Figure 22.3: Accuracy of the fully connected network on the test and training sets

We can see that even simple fully connected network achieves reasonable accuracy, but it has fundamental limitations. It treats each pixel independently, ignoring the spatial structure of the image—the fact that neighboring pixels are related. A fully connected layer connecting a 28×28 image to 1024 hidden units requires over 800,000 parameters, most of which are redundant.

22.3 Convolutions

Convolutional neural networks address these limitations by exploiting two key properties of images:

1. *Local connectivity*: Useful features (edges, textures, shapes) are local—they depend on small neighborhoods of pixels, not the entire image.
2. *Translation invariance*: A feature detector useful in one part of an image is likely useful in other parts.

22.3.1 Key Concepts

A *filter* (also called a *kernel*) is a small matrix of learned weights—typically 3×3 or 5×5 —that extracts local features from an image by computing a weighted sum of pixel values within a sliding window. When the filter’s pattern aligns with a pattern in the image, the dot product produces a high activation; when they don’t match, the activation is low.

- *Kernel size*: The dimensions of the filter (e.g., 3×3). Odd sizes are preferred so the filter has a well-defined center.
- *Receptive field*: The region of the input image that influences a single output value.
- *Stride*: The number of pixels by which the filter shifts at each step. A stride of 1 moves pixel-by-pixel; a stride of 2 skips every other position, reducing the output size.
- *Padding*: Adding zeros around the border of the input to control the output size. “Same” padding preserves the input dimensions; “valid” padding uses no padding, shrinking the output.

While mathematically identical to kernel smoothing in statistics, convolutions in deep learning differ in one critical aspect: the kernel weights are *learned* from data rather than predefined.

Let us look at a one-dimensional example. Suppose we have a one-dimensional input signal x and a one-dimensional filter w . The convolution of x and w is defined as follows:

$$(x * w)(t) = \sum_{i=0}^h x(t+i)w(i),$$

where h is the size of the filter. The convolution operation is used to filter the input signal.

To illustrate this, consider a simple example where we apply a moving average filter to a noisy sinusoidal signal. We generate a sine wave over the interval $[0, 10]$ and add Gaussian noise to simulate realistic measurement errors. The filter uses a uniform window of size 10, where each weight equals $1/10$, effectively computing the local average of the signal. When we convolve this filter with the noisy signal, each output point becomes the weighted average of its neighboring input points, which smooths out the random fluctuations while preserving the underlying periodic structure. Figure 22.4 shows the original noisy signal in light gray and the smoothed result in black, demonstrating how convolution effectively extracts the signal from noise by exploiting local spatial structure.

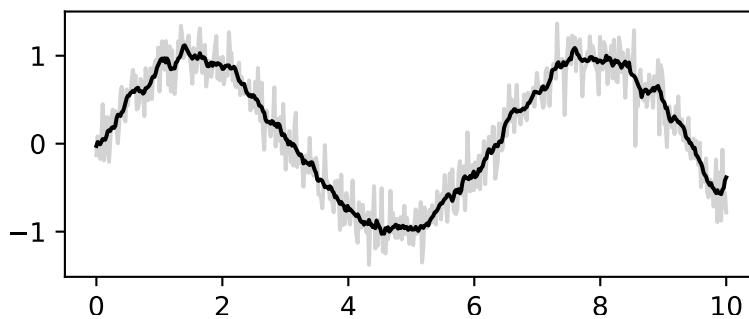
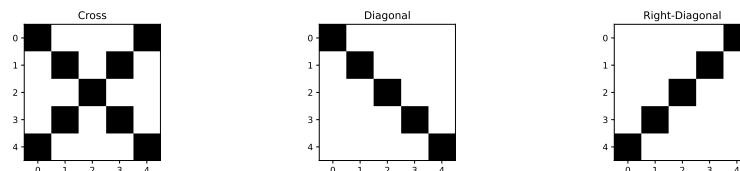


Figure 22.4: Smoothing a noisy signal using 1D convolution with a moving average filter. The light gray line shows the original noisy sinusoidal signal, while the black line displays the smoothed output after applying a uniform filter of size 10.

22.3.2 A Toy Example

To build intuition, consider a simple example with 5×5 images and 3×3 filters. We will classify images into three classes: `cross`, `diagonal`, and `right-diagonal`.



We will use two filters.

The first filter has 1s along its main diagonal and 0s elsewhere—when placed over a matching diagonal line in the image, the dot product is maximized.



The second filter detects the opposite diagonal. This is why convolutions are effective feature detectors: they respond strongly when their pattern matches what's in the image.



We see that both outputs have pixels with high values. However, when we apply the same two filters to the `diagonal` image, we get different results.



The result of applying the second filter to the `diagonal` image is a matrix close to zero—the filter pattern doesn't match the image content. Applying the second filter to the `diagonal` image yields near-zero activations, illustrating that the filter remains dormant when its specific search pattern is absent. This observation motivates the *pooling layer*: by retaining only the maximum activation within each region, the network preserves the presence of a detected feature while discarding its exact location. This provides a degree of translation invariance. Finally, we concatenate the pooled outputs and apply a fully connected layer for classification.

```
## Cross: [0.26163495 0.26163495 0.47673005]
## Diagonal: [0.5937724 0.08035836 0.32586923]
## Right-Diagonal: [0.08035836 0.5937724 0.32586923]
```

The model correctly predicted all three classes. This toy example illustrates the core CNN pipeline: convolutions extract local features, pooling provides spatial invariance, and fully connected layers perform classification.

22.4 CNN for MNIST

Now we apply these concepts to MNIST using **PyTorch**. While we used JAX for the fully connected network to show the low-level mechanics of gradients and matrix operations, we switch to PyTorch here to demonstrate a high-level deep learning framework. PyTorch provides powerful abstractions like `nn.Conv2d` and `nn.Module` that make building and debugging complex architectures like CNNs more intuitive, and it effectively handles the boilerplate of autograd and parameter management.

The architecture follows the same pattern: convolutional layers extract hierarchical features, pooling reduces spatial dimensions, and fully connected layers produce class probabilities.

The CNN uses the Adadelta optimizer, an adaptive learning rate method that adjusts step sizes based on gradient history. Unlike vanilla SGD (which we used for the fully connected network), Adadelta typically requires less hyperparameter tuning.

The CNN architecture is shown in Figure 22.5.

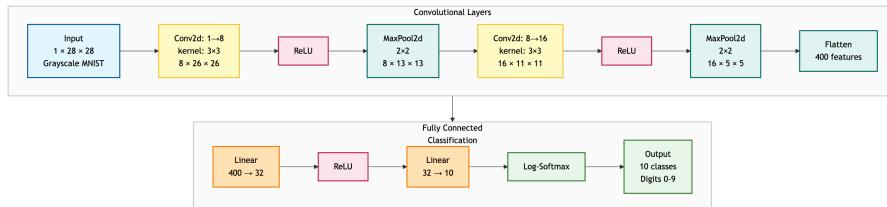


Figure 22.5: CNN Architecture

This model achieves an accuracy of 93% on the test set, compared to 77% for the fully connected network. More importantly, it uses far fewer parameters! The number of parameters in the model is 14k, compared to 800k for the fully connected network. The convolutional layers share weights across spatial positions rather than learning separate weights for each pixel location. This parameter efficiency, combined with the inductive bias toward local and translation-invariant features, explains why CNNs have become the dominant architecture for computer vision tasks.

22.5 Summary

Convolutional neural networks exploit the spatial structure of images through three key mechanisms:

1. *Convolutional layers* apply learned filters that detect local patterns, sharing weights across the entire image.
2. *Pooling layers* reduce spatial dimensions while providing invariance to small translations.
3. *Hierarchical feature learning*: early layers detect simple features (edges, textures); deeper layers combine these into complex patterns (shapes, objects).

The CNN architecture we built achieves significantly higher accuracy than the fully connected network while using far fewer parameters. This parameter efficiency, combined with the inductive bias toward local and translation-invariant features, explains why CNNs have become the dominant architecture for computer vision tasks.

23

Natural Language Processing

The ability to understand and generate human language has long been considered a hallmark of intelligence. When Alan Turing proposed his famous test in 1950, he chose natural conversation as the ultimate benchmark for machine intelligence. Yet for decades, this goal remained frustratingly elusive. Early attempts at machine translation in the 1950s, which simply replaced words using bilingual dictionaries, produced nonsensical results that highlighted the profound complexity of human language. The phrase “The spirit is willing, but the flesh is weak” allegedly translated to Russian and back as “The vodka is good, but the meat is rotten”—a cautionary tale about the subtleties of meaning that transcend mere word substitution.

This chapter traces the remarkable journey from those early failures to today’s language models that can engage in nuanced dialogue, translate between languages with near-human accuracy, and even generate creative text. At the heart of this transformation lies a fundamental shift in how we represent language computationally: from discrete symbols manipulated by hand-crafted rules to continuous vector spaces learned from vast corpora of text.

The chapter develops the mathematical foundations of natural language understanding: *word embeddings*, *attention mechanisms*, and the *Transformer* architecture that powers modern language models like BERT and GPT.

23.1 Converting Words to Numbers (Embeddings)

Language poses distinctive challenges for mathematical modeling: unlike images, which naturally exist as arrays of continuous pixel values, or audio signals, which are continuous waveforms, text consists of discrete symbols with no inherent geometric structure. The word “cat” is not inherently closer to “dog” than to “quantum”—at least not in any obvious mathematical sense. Yet humans effortlessly recognize that cats and dogs share semantic properties that neither shares with abstract physics concepts.

A naive way to represent words is through one-hot encoding, where each word

in a vocabulary is assigned a unique vector with a single non-zero entry. For example, in a vocabulary of size $|V|$, the word “cat” might be represented as

$$v_{\text{cat}} = \overbrace{[0, 0, \dots, 1, \dots, 0]}^{|V|}$$

with the 1 in the i -th position corresponding to “cat”. However, this approach fails to capture any notion of semantic similarity: the cosine similarity between any two distinct one-hot vectors is zero, erasing all information about how words relate to each other.

This type of representation makes even the seemingly simple task of determining whether two sentences have similar meanings challenging. The sentences “The cat sat on the mat” and “A feline rested on the rug” express nearly identical ideas despite sharing no words except “the” and “on.” Conversely, “The bank is closed” could refer to a financial institution or a river’s edge—the same words encoding entirely different meanings. These examples illustrate why early symbolic approaches to natural language processing, based on logical rules and hand-crafted features, struggled to capture the fluid, contextual nature of meaning.

The breakthrough came from reconceptualizing the representation problem. Instead of treating words as atomic symbols, we embed them in a continuous vector space where geometric relationships encode semantic relationships. In such a space, we might find that $v_{\text{cat}} - v_{\text{dog}}$ has similar direction to $v_{\text{car}} - v_{\text{bicycle}}$, capturing analogical relationships through vector arithmetic.

To formalize this intuition, we seek a mapping $\phi : \mathcal{V} \rightarrow \mathbb{R}^d$ from a vocabulary \mathcal{V} of discrete tokens to d -dimensional vectors. The challenge lies in learning this mapping such that the resulting geometry reflects semantic relationships.

23.1.1 The Math of Twenty Questions

My (Vadim’s) daughter and I play a game of Twenty Questions during road trips. The rules are simple: one person thinks of something, and the other person has to guess what it is by asking yes-or-no questions. The person who is guessing can ask up to twenty questions, and then they have to make a guess. If they guess correctly, they win; if not, the other person wins. The game is fun, but it’s also a great way to illustrate how AI systems can learn to represent words and phrases as numbers. The trick is to come up with an optimal set of yes-or-no questions that will allow you to distinguish between all the words or phrases you might want to represent. Surprisingly, most of the words can be identified with a small set of universal questions asked in the same order every time. Usually the person who has a better set of questions wins. For example, you might ask:

- Is it an animal? (Yes)

- Is this a domestic animal? (No)
- Is it larger than a human? (Yes)
- Does it have a long tail? (No)
- Is it a predator? (Yes)
- Can move on two feet? (Yes)
- Is it a bear? (Yes)

Thus, if we use a 20-dimensional 0-1 vector to represent the word “bear,” the portion of this vector corresponding to these questions would look like this:

	Larger than		Long	Can move on		
	Animal	Domestic	human	tail	Predator	two feet
Bear	1	0	1	0	1	1

This is called a word vector. Specifically, it’s a “binary” or 0/1 vector: 1 means yes, 0 means no. Different words would produce different answers to the same questions, so they would have different word vectors. If we stack all these vectors in a matrix, where each row is a word and each column is a question, we get something like this:

	Larger than		Long	Can move on		
	Animal	Domestic	human	tail	Predator	two feet
Bear	1	0	1	0	1	1
Dog	1	1	0	1	0	0
Cat	1	1	0	1	1	0

The binary nature of these vectors forces us to choose 1 or 0 for the “Larger than human” question for “Dog.” However, there are some dog breeds that are larger than humans, so this binary representation is not very useful in this case. We can do better by allowing the answers to be numbers between 0 and 1, rather than just 0 or 1. This way, we can represent the fact that some dogs are larger than humans, but most are not. For example, we might answer the question “Is it larger than a human?” with a 0.1 for a dog and a 0.8 for a bear. Some types of bears can be smaller than humans, for example, black bears that live in North America, but most bears are larger than humans.

Using this approach, the vectors now become

	Larger than		Long	Can move on		
	Animal	Domestic	human	tail	Predator	two feet
Bear	1	0	0.8	0	1	0.8
Dog	1	1	0.1	0.6	0	0

	Larger than Animal Domestic human		Long tail	Predator	Can move on two feet
Cat	1	0.7	0.01	1	0.6

AI systems also use non-binary scoring rules to judge a win or loss. For example, if the answer is “bear,” then the score might be 100 points for a correct guess, 90 points for a close guess like “wolf” or “lion,” and 50 points for a distant guess like “eagle.” This way of keeping score matches the real-world design requirements of most NLP systems. For example, if you translate JFK saying “Ich bin ein Berliner” as “I am a German,” you’re wrong, but a lot closer than if you translate it as “I am a butterfly.”

The process of converting words into numbers is called “embedding.” The resulting vectors are called “word embeddings.” The only part that is left is how to design an algorithm that can find a good set of questions to ask. Usually real-life word embeddings have hundreds of questions, not just twenty. The process of finding these questions is called “training” the model. The goal is to find a set of questions that will allow the model to distinguish between all the words in the vocabulary, and to do so in a way that captures their meanings. However, the algorithms do not have a notion of meaning in the same way that humans do. Instead, they learn by counting word co-location statistics—that is, which words tend to appear with which other words in real sentences written by humans.

These co-occurrence statistics serve as surprisingly effective proxies for meaning. For example, consider the question: “Among all sentences containing ‘fries’ and ‘ketchup,’ how frequently does the word ‘bun’ also appear?” This is the kind of query a machine can easily formulate and answer, since it relies on counting rather than true understanding.

While such a specific question may be too limited if you can only ask a few hundred, the underlying idea—using word co-occurrence patterns—is powerful. Word embedding algorithms are built on this principle: they systematically explore which words tend to appear together, and through optimization, learn the most informative “questions” to ask. By repeatedly applying these learned probes, the algorithms construct a vector for each word or phrase, capturing its relationships based on co-occurrence statistics. These word vectors are then organized into a matrix for further use.

There are many ways to train word embeddings, but the most common one is to use a neural network. The neural network learns to ask questions that are most useful for distinguishing between different words. It does this by looking at a large number of examples of words and their meanings, and then adjusting the weights of the questions based on how well they perform. One of the first and most popular algorithms for this is called Word2Vec, which was introduced by Mikolov et al. (2013) at Google in 2013.

23.2 Word2Vec and Distributional Semantics

The theoretical foundation for learning meaningful word representations comes from the distributional hypothesis, articulated by linguist J.R. Firth in 1957: “You shall know a word by the company it keeps.” This principle suggests that words appearing in similar contexts tend to have similar meanings. If “coffee” and “tea” both frequently appear near words like “drink,” “hot,” “cup,” and “morning,” we can infer their semantic similarity.

The word2vec framework, introduced by Mikolov et al. (2013), operationalized this insight through a probabilistic model. The skip-gram variant posits that a word can be used to predict its surrounding context words. Given a corpus of text represented as a sequence of words w_1, w_2, \dots, w_T , the model maximizes the likelihood:

$$\mathcal{L} = \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} | w_t)$$

where m is the context window size. The conditional probability is parameterized using two sets of embeddings: \mathbf{v}_w for words as centers and \mathbf{u}_w for words as context:

$$P(w_o | w_c) = \frac{\exp(\mathbf{u}_o^T \mathbf{v}_c)}{\sum_{w \in \mathcal{V}} \exp(\mathbf{u}_w^T \mathbf{v}_c)}$$

This formulation reveals deep connections to the theoretical frameworks discussed in previous chapters. The dot product $\mathbf{u}_o^T \mathbf{v}_c$ acts as a compatibility score between center and context words, while the softmax normalization ensures a valid probability distribution. From the perspective of ridge functions, we can view this as learning representations where the function $f(w_c, w_o) = \mathbf{u}_o^T \mathbf{v}_c$ captures the log-odds of co-occurrence.

23.3 Word2Vec for War and Peace

This analysis showcases the application of Word2Vec on Leo Tolstoy’s “War and Peace,” demonstrating how neural network-based techniques can learn vector representations of words by analyzing their contextual relationships within the text. The model employs the skip-gram algorithm with negative

sampling (explained later), a widely used configuration for Word2Vec. We use vector dimension of 100, providing a balance between capturing semantic detail and computational efficiency. The context window size is set to 5, and the model is trained over several iterations to ensure convergence. Finally, we use PCA to reduce the dimensionality of the vectors to 2D for visualization.

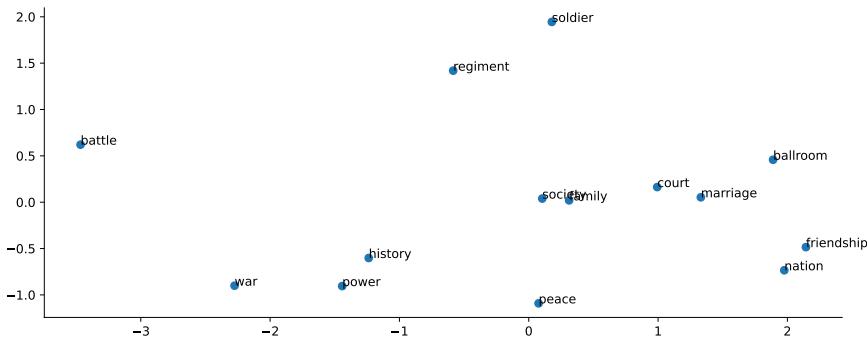


Figure 23.1: Word2Vec for War and Peace

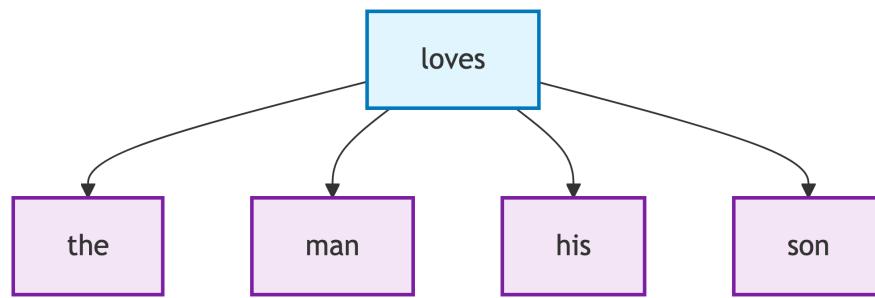
The results reveal meaningful semantic relationships within the text and highlight several key insights into the thematic structure of Tolstoy's War and Peace. First, the clustering of military-related terms (**soldier**, **regiment**, **battle**) underscores Word2Vec's ability to group semantically related words based on their co-occurrence patterns. Additionally, the grouping of words such as "ballroom," "court," and "marriage" reflects the novel's emphasis on aristocratic society and its social hierarchy. The use of PCA for dimensionality reduction effectively preserves meaningful relationships while reducing the original high-dimensional Word2Vec space (100 dimensions) to a two-dimensional representation. Overall, this visualization demonstrates how Word2Vec captures distinct thematic domains, including military, social, and government. Notice that a more abstract concept like **peace** is surrounded by words from both government (**history**, **power**, **war**) and social domains (**family**, **friendship**, **marriage**, **court**, **ballroom**, **society**), indicating its central role in the narrative but is distant from the military domain, which is more focused on the war aspect of the story.

These insights have practical applications in literary analysis, theme extraction, character relationship mapping, historical text understanding, and similarity-based text search and recommendation systems.

23.3.1 The Skip-Gram Model

The skip-gram model predicts surrounding context words within a fixed window given a center word. This approach assumes that words appearing in similar contexts tend to have similar meanings, directly implementing the distributional hypothesis.

Consider the sentence “The man loves his son” with “loves” as the center word and a context window of size 2. The skip-gram model aims to maximize the probability of generating the context words “the,” “man,” “his,” and “son” given the center word “loves.” This relationship can be visualized as follows:



The mathematical foundation assumes conditional independence among context words given the center word, allowing the joint probability to factorize:

$$\begin{aligned} P(\text{"the"}, \text{"man"}, \text{"his"}, \text{"son"} \mid \text{"loves"}) &= \\ P(\text{"the"} \mid \text{"loves"}) \cdot P(\text{"man"} \mid \text{"loves"}) \cdot P(\text{"his"} \mid \text{"loves"}) \cdot P(\text{"son"} \mid \text{"loves"}) \end{aligned}$$

This independence assumption, while not strictly true in natural language, proves crucial for computational tractability. Each conditional probability is modeled using the softmax function over the entire vocabulary:

$$P(w_o \mid w_c) = \frac{\exp(\mathbf{u}_o^T \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^T \mathbf{v}_c)}$$

The skip-gram objective seeks to maximize the likelihood of observing all context words across the entire corpus. For a text sequence of length T with words $w^{(1)}, w^{(2)}, \dots, w^{(T)}$, the objective becomes:

$$\mathcal{L}_{\text{skip-gram}} = \frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w^{(t+j)} \mid w^{(t)})$$

where m is the context window size. The normalization by T ensures that the objective remains bounded as corpus size grows.

The gradient with respect to the center word embedding reveals the learning dynamics:

$$\frac{\partial \log P(w_o | w_c)}{\partial \mathbf{v}_c} = \mathbf{u}_o - \sum_{i \in \mathcal{V}} P(w_i | w_c) \mathbf{u}_i$$

The gradient pushes the center word embedding toward the observed context word (\mathbf{u}_o) while pulling it away from all other words, weighted by their predicted probabilities. This creates a natural contrast between positive and negative examples.

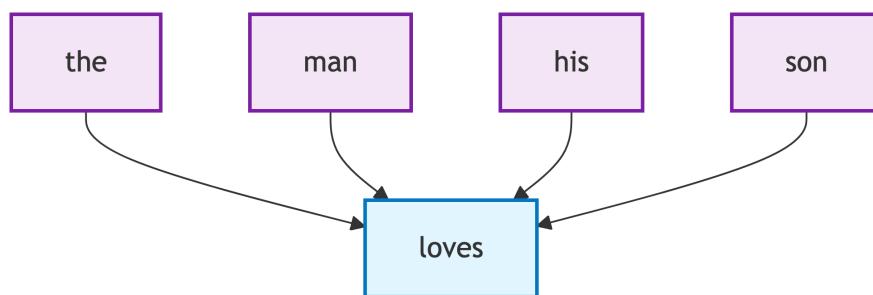
The skip-gram architecture assigns two vector representations to each word: \mathbf{v}_w when the word serves as a center word and \mathbf{u}_w when it appears in context. This asymmetry allows the model to capture different aspects of word usage. After training, the center word vectors \mathbf{v}_w are typically used as the final word embeddings, though some implementations average or concatenate both representations.

23.3.2 The Continuous Bag of Words (CBOW) Model

While skip-gram predicts context words from a center word, the Continuous Bag of Words (CBOW) model reverses this relationship: it predicts a center word based on its surrounding context. For the same text sequence “the”, “man”, “loves”, “his”, “son” with “loves” as the center word, CBOW models the conditional probability:

$$P(\text{"loves"} | \text{"the"}, \text{"man"}, \text{"his"}, \text{"son"})$$

The CBOW architecture can be visualized as multiple context words converging to predict a single center word:



The key difference from skip-gram lies in how CBOW handles multiple context words. Rather than treating each context word independently, CBOW averages their embeddings. For a center word w_c with context words $w_{o_1}, \dots, w_{o_{2m}}$, the conditional probability is:

$$P(w_c | w_{o_1}, \dots, w_{o_{2m}}) = \frac{\exp(\mathbf{u}_c^T \bar{\mathbf{v}}_o)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^T \bar{\mathbf{v}}_o)}$$

where $\bar{\mathbf{v}}_o = \frac{1}{2m} \sum_{j=1}^{2m} \mathbf{v}_{o_j}$ is the average of context word vectors. Note that in CBOW, \mathbf{u}_i represents word i as a center word and \mathbf{v}_i represents it as a context word—the opposite of skip-gram’s convention.

The CBOW objective maximizes the likelihood of generating all center words given their contexts:

$$\mathcal{L}_{\text{CBOW}} = \sum_{t=1}^T \log P(w^{(t)} | w^{(t-m)}, \dots, w^{(t-1)}, w^{(t+1)}, \dots, w^{(t+m)})$$

The gradient with respect to context word vectors reveals how CBOW learns:

$$\frac{\partial \log P(w_c | \mathcal{W}_o)}{\partial \mathbf{v}_{o_i}} = \frac{1}{2m} \left(\mathbf{u}_c - \sum_{j \in \mathcal{V}} P(w_j | \mathcal{W}_o) \mathbf{u}_j \right)$$

This gradient is scaled by $\frac{1}{2m}$, effectively distributing the learning signal across all context words. CBOW tends to train faster than skip-gram because it predicts one word per context window rather than multiple words, but skip-gram often produces better representations for rare words since it generates more training examples per sentence.

23.3.3 Pretraining Word2Vec

Training word2vec models requires careful attention to implementation details that significantly impact the quality of learned representations. The training process begins with data preprocessing on large text corpora. Using the Penn Tree Bank dataset as an example—a carefully annotated corpus of Wall Street Journal articles containing about 1 million words—we implement several crucial preprocessing steps.

First, we build a vocabulary by counting word frequencies and retaining only words that appear at least a minimum number of times (typically 5-10). This thresholding serves two purposes: it reduces the vocabulary size from potentially millions to tens of thousands of words, and it prevents the model from

wasting capacity on rare words that appear too infrequently to learn meaningful representations. Words below the threshold are replaced with a special `<unk>` token.

The training procedure uses stochastic gradient descent with a carefully designed learning rate schedule. The initial learning rate (typically 0.025 for skip-gram and 0.05 for CBOW) is linearly decreased to a minimum value (usually 0.0001) as training progresses:

$$\alpha_t = \alpha_0 \left(1 - \frac{\text{words_processed}}{\text{total_words} \times \text{epochs}} \right)$$

This schedule ensures larger updates early in training when representations are poor, transitioning to fine-tuning as the model converges.

The implementation uses several optimizations for efficiency. Word vectors are typically initialized randomly from a uniform distribution over $[-0.5/d, 0.5/d]$ where d is the embedding dimension (commonly 100-300). During training, we maintain two embedding matrices: one for center words and one for context words. After training, these can be combined (usually by averaging) or just the center word embeddings can be used.

For minibatch processing, training examples are grouped to enable efficient matrix operations. Given a batch of center-context pairs, the forward pass computes scores using matrix multiplication, applies the loss function (either full softmax, hierarchical softmax, or negative sampling), and backpropagates gradients. A typical training configuration might process batches of 512 word pairs, iterating 5-15 times over a corpus.

The quality of learned embeddings can be evaluated through word similarity and analogy tasks. For similarity, we compute cosine distances between word vectors and verify that semantically similar words have high cosine similarity. For analogies, we test whether vector arithmetic captures semantic relationships: the famous “king - man + woman \approx queen” example demonstrates that vector differences encode meaningful semantic transformations.

A crucial but often overlooked aspect is subsampling of frequent words. Words like “the,” “a,” and “is” appear so frequently that they provide little information about the semantic content of their neighbors. The probability of discarding a word w during training is:

$$P(\text{discard } w) = 1 - \sqrt{\frac{t}{f(w)}}$$

where $f(w)$ is the frequency of word w and t is a threshold (typically 10^{-5}). This formula ensures that very frequent words are aggressively subsampled while preserving most occurrences of informative words.

For training, we extract examples by sliding a window over the text. For each center word, we collect its context words within a window of size m . Importantly, the actual window size is sampled uniformly from $[1, m]$ for each center word, which helps the model learn representations that are robust to varying context sizes. Consider the sentence “the cat sat on the mat” with maximum window size 2. For the center word “sat,” we might sample a window size of 1, giving context words [“cat”, “on”], or a window size of 2, giving context words [“the”, “cat”, “on”, “the”].

23.4 Computational Efficiency Through Negative Sampling

The elegance of word2vec’s formulation belies a serious computational challenge. Computing the normalization term in the softmax requires summing over the entire vocabulary—potentially millions of terms—for every gradient update. With large corpora containing billions of words, this quickly becomes intractable.

Negative sampling transforms the problem from multi-class classification to binary classification. Instead of predicting which word from the entire vocabulary appears in the context, we ask a simpler question: given a word pair, is it a real center-context pair from the corpus or a randomly generated negative example? The objective becomes:

$$\mathcal{L}_{\text{NS}} = \log \sigma(\mathbf{u}_o^T \mathbf{v}_c) + \sum_{k=1}^K \mathbb{E}_{w_k \sim P_n} [\log \sigma(-\mathbf{u}_{w_k}^T \mathbf{v}_c)]$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function, and P_n is a noise distribution over words. The clever insight is that by carefully choosing the noise distribution—typically $P_n(w) \propto f(w)^{3/4}$ where $f(w)$ is word frequency—we can approximate the original objective while reducing computation from $O(|\mathcal{V}|)$ to $O(K)$, where $K \ll |\mathcal{V}|$ is a small number of negative samples (typically 5-20).

An alternative solution is hierarchical softmax, which replaces the flat softmax over the vocabulary with a binary tree where each leaf represents a word. The probability of a word is then the product of binary decisions along the path from root to leaf:

$$P(w_o | w_c) = \prod_{j=1}^{L(w_o)-1} \sigma([\![n(w_o, j+1) = \text{leftChild}(n(w_o, j))]\!] \cdot \mathbf{u}_{n(w_o, j)}^T \mathbf{v}_c)$$

where $L(w_o)$ is the length of the path to word w_o , $n(w_o, j)$ is the j -th node on this path, and $\llbracket \cdot \rrbracket$ is the Iverson bracket—a notation that returns $+1$ if the condition inside is true (the next node is the left child) and -1 otherwise (the next node is the right child). This sign flip determines whether we use the sigmoid or its complement at each binary decision. Hierarchical softmax reduces computational complexity from $O(|\mathcal{V}|)$ to $O(\log |\mathcal{V}|)$, though negative sampling typically performs better in practice.

23.5 Global Vectors and Matrix Factorization

While word2vec learns from local context windows, GloVe (Global Vectors) leverages global co-occurrence statistics. The key observation is that the ratio of co-occurrence probabilities can encode semantic relationships. Consider the words “ice” and “steam” in relation to “solid” and “gas”: $P(\text{solid} | \text{ice})/P(\text{solid} | \text{steam})$ is large (around 8.9), while $P(\text{gas} | \text{ice})/P(\text{gas} | \text{steam})$ is small (around 0.085), and $P(\text{water} | \text{ice})/P(\text{water} | \text{steam})$ is close to 1 (around 1.36).

These ratios capture the semantic relationships: ice is solid, steam is gas, and both relate to water. GloVe learns embeddings that preserve these ratios through the objective:

$$\mathcal{L}_{\text{GloVe}} = \sum_{i,j} h(X_{ij}) (\mathbf{v}_i^T \mathbf{u}_j + b_i + c_j - \log X_{ij})^2$$

where X_{ij} counts co-occurrences, $h(\cdot)$ is a weighting function that prevents very common or very rare pairs from dominating, and b_i, c_j are bias terms. A typical choice is $h(x) = (x/x_{\max})^\alpha$ if $x < x_{\max}$, else 1, with $\alpha = 0.75$ and $x_{\max} = 100$.

This formulation reveals GloVe as a weighted matrix factorization problem. We seek low-rank factors \mathbf{V} and \mathbf{U} such that $\mathbf{V}^T \mathbf{U} \approx \log \mathbf{X}$, where the approximation is weighted by the function $h(\cdot)$. This connection to classical linear algebra provides theoretical insights: the optimal embeddings lie in the subspace spanned by the top singular vectors of an appropriately transformed co-occurrence matrix.

23.6 Beyond Words: Subword and Character Models (Tokenization)

A fundamental limitation of word-level embeddings is their inability to handle out-of-vocabulary words or capture morphological relationships. The word “unhappiness” shares obvious morphological connections with “happy,” “unhappy,” and “happiness,” but word2vec treats these as completely independent tokens.

FastText addresses this limitation by representing words as bags of character n-grams. For the word “where” with n-grams of length 3 to 6, we extract: “<wh”, “whe”, “her”, “ere”, “re>”, and longer n-grams up to the full word. The word embedding is then the sum of its n-gram embeddings:

$$\mathbf{v}_{\text{where}} = \sum_{g \in \mathcal{G}_{\text{where}}} \mathbf{z}_g$$

This approach handles out-of-vocabulary words by breaking them into known n-grams and provides better representations for rare words by sharing parameters across morphologically related words.

An even more flexible approach is Byte Pair Encoding (BPE), which learns a vocabulary of subword units directly from the data. Starting with individual characters, BPE iteratively merges the most frequent pair of adjacent units until reaching a desired vocabulary size. For example, given a corpus with word frequencies {“fast”: 4, “faster”: 3, “tall”: 5, “taller”: 4}, BPE might learn merges like “t” + “a” → “ta”, then “ta” + “l” → “tal”, and so on. This data-driven approach balances vocabulary size with representation power, enabling models to handle arbitrary text while maintaining reasonable computational requirements.

The choice of tokenization strategy has profound implications for model performance and behavior. Modern LLMs typically use vocabularies of 50,000 to 100,000 tokens, carefully balanced to represent different languages while maintaining computational efficiency. A vocabulary that’s too small forces the model to represent complex concepts with many tokens, making it harder to capture meaning efficiently.

Consider how “The Verbasizer helped Bowie create unexpected word combinations” might be tokenized: [“The”, “Ver”, “bas”, “izer”, “helped”, “Bow”, “ie”, “create”, “unexpected”, “word”, “combinations”]. This fragmentation handles rare words and proper names by breaking them into common subcomponents.

This tokenization process explains some behavioral patterns in modern language models. Names from fictional universes, specialized terminology, and code snippets can be tokenized in ways that fragment their semantic meaning, potentially affecting model performance.

23.7 Attention Mechanisms and Contextual Representations

Static word embeddings suffer from a fundamental limitation: they assign a single vector to each word, ignoring context. The word “bank” receives the same representation whether it appears in “river bank” or “investment bank.” This conflation of multiple senses into a single vector creates an information bottleneck that limits performance on downstream tasks.

The evolution of word embeddings culminated in contextual representations, which dynamically adjust based on surrounding context. The context problem was solved by attention mechanisms, particularly the transformer architecture, which allows models to capture complex dependencies between words. Attention mimics the human ability to selectively focus: when reading, your visual system processes every word simultaneously, yet conscious attention flows sequentially. You can redirect this attention at will—noticing a particular phrase or jumping back to reread a complex clause.

Machine learning systems faced analogous challenges in processing sequential information. Early neural networks, particularly recurrent architectures, processed sequences step by step, maintaining a hidden state that theoretically encoded all previous information. However, this sequential bottleneck created fundamental limitations: information from early time steps could vanish as sequences grew longer, and the inherently sequential nature prevented parallel computation that could leverage modern hardware effectively.

The attention Vaswani et al. (2023) mechanism enabled this landscape by allowing models to directly access and combine information from any part of the input sequence. Rather than compressing an entire sequence into a fixed-size vector, attention mechanisms create dynamic representations that adaptively weight different parts of the input based on their relevance to the current computation. This breakthrough enabled the development of the Transformer architecture, which forms the foundation of modern language models from BERT to GPT.

At its core, attention implements an information retrieval system inspired by database operations. Consider how you might search through a library: you formulate a **query** (what you’re looking for), examine the **keys** (catalog

entries or book titles), and retrieve the associated **values** (the actual books or their contents). Attention mechanisms formalize this intuition through three learned representations:

- *Queries (Q)*: What information are we seeking?
- *Keys (K)*: What information is available at each position?
- *Values (V)*: What information should we retrieve?

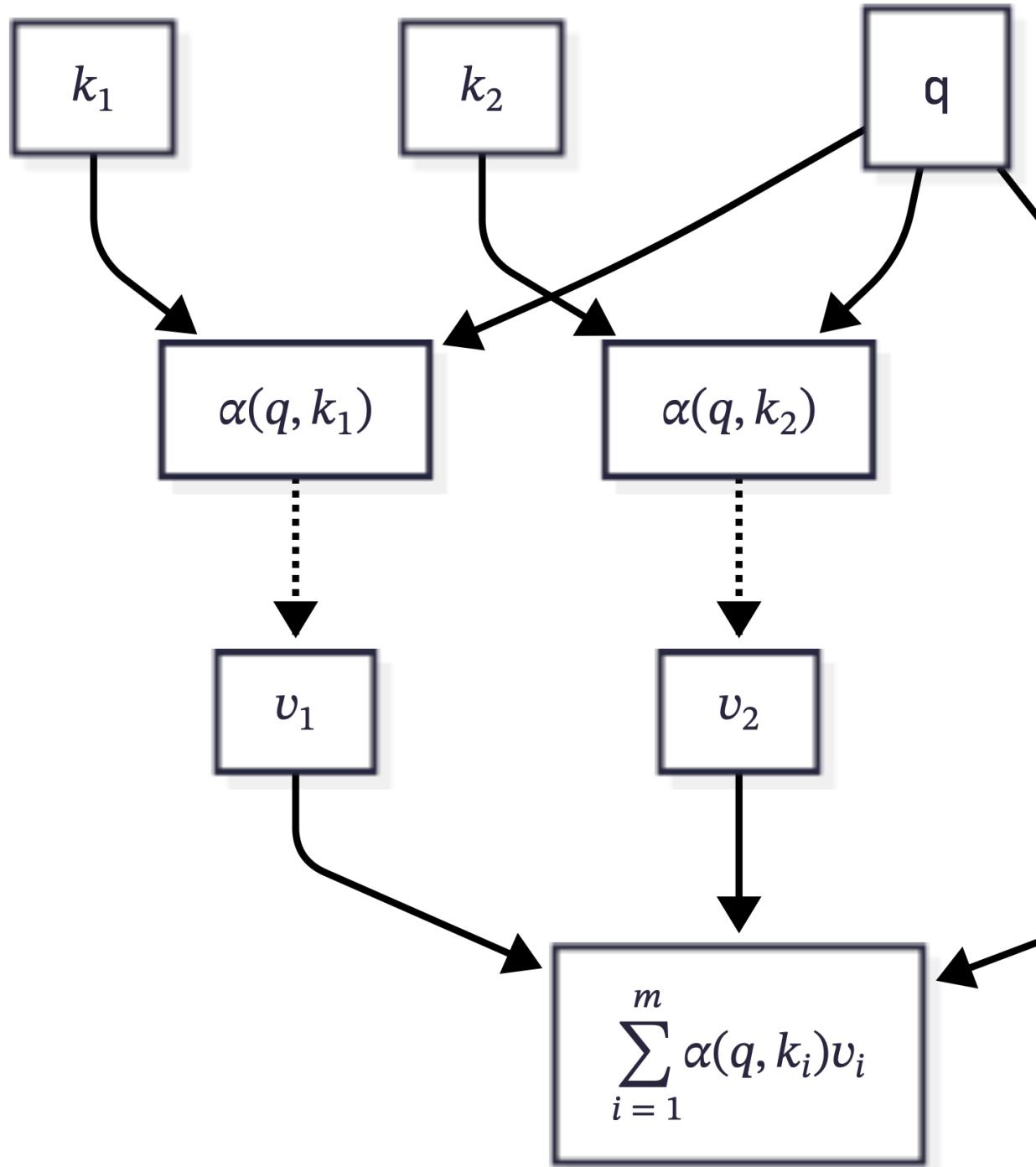
Given a set of keys and values, attention computes a weighted sum of the values based on their relevance to the query. Mathematically, this can be expressed as:

$$\text{Attention}(\mathbf{q}) \stackrel{\text{def}}{=} \sum_{i=1}^m \alpha_i(\mathbf{q}, \mathbf{k}_i) \mathbf{v}_i,$$

where α_i are scalar attention weights computed from the queries and keys. The operation itself is typically referred to as *attention pooling*. The name “attention” derives from the fact that the operation pays particular attention to the terms for which the weight α_i is significant (i.e., large).

It is typical in AI applications to assume that the attention weights α_i are nonnegative and form a convex combination, i.e., $\sum_{i=1}^n \alpha_i = 1$. A common strategy for ensuring that the weights sum up to 1 is to normalize the unnormalized scores s_i via the softmax function:

$$\alpha_i = \frac{\exp(s_i)}{\sum_{j=1}^n \exp(s_j)}$$



This diagram shows the basic attention computation. A query vector q is compared to each key k_i to produce a score, which is then turned into an attention weight $\alpha(q, k_i)$ (often normalized so the weights sum to one). The output is a weighted sum of the value vectors, $\sum_{i=1}^m \alpha(q, k_i)v_i$, so values associated with keys most relevant to the query contribute more to the final representation.

23.8 Kernel Smoothing as Attention

The concept of attention pooling can actually be traced back to classical kernel methods like Nadaraya-Watson regression, where similarity kernels determine how much weight to give to different data points. Given a sequence of observations y_1, \dots, y_n , and each observation has a two-dimensional index $x^{(i)} \in \mathbb{R}^2$, such as in a spatial process. Then the question is what would be the value of y at a previously unobserved location $x_{new} = (x_1, x_2)$. Let's simulate some data and see what we can do.

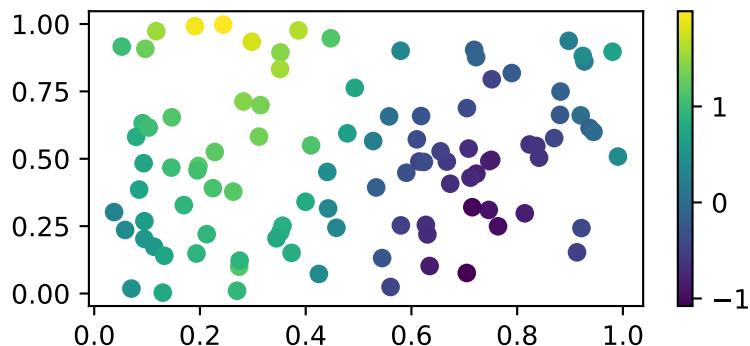


Figure 23.2: Simulated data for kernel smoothing

The kernel smoothing method then uses $q = (x_1, x_2)$ as the query and observed x_i s as the keys. The weights are then computed as $\alpha_i = K(q, x_i)$ where K is a kernel function, that measures, how close q is to x_i . Then, our prediction would be

$$\hat{y} = \sum_{i=1}^n \alpha_i y_i.$$

The weights α_i assign importance to y_i based on how close $q = (x_1, x_2)$ is to $x^{(i)}$. Common kernel choices and their formulas are summarized below:

Kernel	Formula for $K(q, x^{(i)})$
Gaussian	$\exp\left(-\frac{\ q-x^{(i)}\ ^2}{2h^2}\right)$
Boxcar	$\mathbb{I}(\ q-x^{(i)}\ < h)$
Constant	1
Epanechnikov	$\max\left(1 - \frac{\ q-x^{(i)}\ }{h}, 0\right)$

Here, h is a bandwidth parameter controlling the width of the kernel, and \mathbb{I} is the indicator function. Code for the kernels is given below.

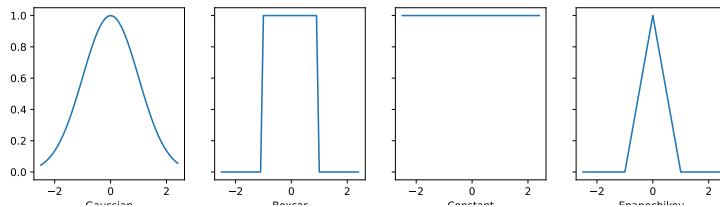


Figure 23.3: Plotting kernels

Each of these kernels represents a different way of weighting information based on similarity or distance. In neural networks, this translates to learning how to attend to different parts of the input sequence, but with the flexibility to learn much more complex patterns than these simple mathematical functions.

Now, let's plot the true value of the function and the kernel smoothed value.

```
## <matplotlib.colorbar.Colorbar object at 0x11e1f2f60>
## <matplotlib.colorbar.Colorbar object at 0x11e228c50>
## <matplotlib.colorbar.Colorbar object at 0x11e2cf920>
```

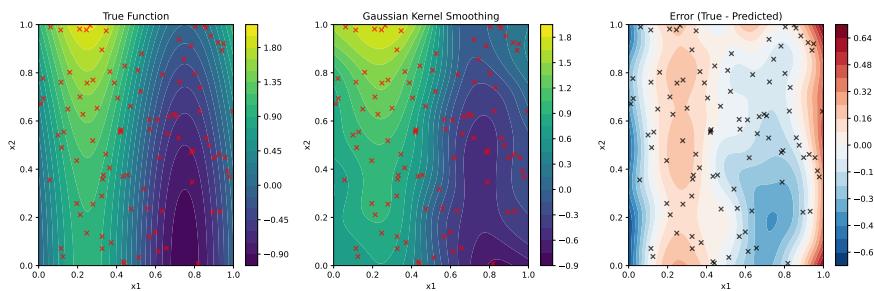


Figure 23.4: Kernel Smoothing using Gaussian Kernel

We can see that a relatively simple concept of calculating a prediction by averaging the values of the training data, weighted by the kernel function, can be used to estimate the value of the function at a new point rather well. Clearly, the model is not performing well in the regions where we do not have training data, but it is still able to capture the general shape of the function.

23.8.1 Attention over a sequence of words

This mechanism liberates the model from sequential processing. It can process an entire sequence at once, draw connections between distant words, and process information in parallel.

There are two variations of attention that are particularly important: **Cross-attention** allows a model to weigh the importance of words in an input sequence (like an encoder's output) when generating a new output sequence (in the decoder). **Self-attention** allows a model to weigh the importance of different words within a single input sequence.

In self-attention, queries, keys, and values all come from the same sequence. This allows each position in the sequence to attend to all other positions, creating rich representations that capture the complex web of relationships within a single piece of text. When you read a sentence like “The trophy would not fit in the brown suitcase because it was too big,” self-attention helps the model understand that “it” most likely refers to the trophy, not the suitcase, by considering the relationships between all the words simultaneously.

Let's now consider a sequence of words (or tokens), where each word is represented as a vector (embedding)

$$H = [h_1, h_2, \dots, h_n] \in \mathbb{R}^{n \times d}.$$

Here n is the length of our sequence to be analyzed. It can be a sentence, a paragraph, a document. In case of self-attention, the trio of Q, K, V are all projections of the same input sequence H

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V.$$

The projections are done by multiplying the input sequence H by a learnable parameter matrix $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_k}$. Thus the resulting Q, K, V are of dimension $n \times d_k$. Now, we can calculate the unnormalized attention scores as:

$$s_{ij} = \frac{q_i^T k_j}{\sqrt{d_k}}, \quad i, j = 1, \dots, n.$$

where q_i is the i -th row of Q , k_j is the j -th row of K , and d_k is the dimension of the key vectors. The attention weights are then obtained by normalizing

with the softmax function:

$$\alpha_{ij} = \frac{\exp(s_{ij})}{\sum_{k=1}^n \exp(s_{ik})}.$$

This softmax normalization ensures that attention weights sum to one, creating a proper probability distribution over the input positions. The scaling factor $1/\sqrt{d_k}$ prevents the dot products from becoming too large, which would push the softmax into regions with extremely small gradients.

The final output is then computed as:

$$o_i = \sum_{j=1}^n \alpha_{ij} v_j.$$

where o_i is the i -th row of the output.

Notice that instead of using the Gaussian kernel, as we did before, we simply use the dot product. This makes computation much faster and more efficient, while still being able to capture the similarity between the words. Matrix-matrix product operation is implemented in all major deep learning libraries. It is differentiable, ensuring that its gradient remains stable, which is crucial for model training. Alternative attention mechanisms exist—notably non-differentiable variants trained via reinforcement learning (Mnih et al. 2014)—but their training complexity has led most researchers to adopt the differentiable framework we present here.

The dot product naturally measures the alignment or “similarity” between two vectors. A larger dot product implies the vectors are pointing in a similar direction. When combined with the scaling factor and the softmax function, this simple calculation has proven to be a highly effective way to determine which tokens are most relevant to each other for language processing tasks. It provides a good signal for “unnormalized alignment” before being converted into a probability distribution. In fact, the dot product is a special case of a kernel function. For normalized vectors, the dot product is equivalent to the cosine similarity.

$$Q_i \cdot K_j = \cos(\theta_{ij})$$

where θ_{ij} is the angle between the i -th and j -th vectors. It is proportional to

$$\cos(\theta_{ij}) \propto -\|Q_i - K_j\|^2$$

up to constants. In fact, if Q and K are normalized and σ is tuned, the Gaussian kernel becomes almost equivalent to a softmax over dot products. For the specific tasks that attention methods excel at, more complex kernels like the Gaussian kernel haven’t demonstrated a consistent or significant enough improvement in performance to justify their increased computational overhead.

Now instead of calculating norms, we calculate the dot product and instead of having a bandwidth parameter *that needs to be tuned*, we have projection matrices W_Q, W_K, W_V , that are learned during training.

For example, let's consider the following sentence:

“The cat sat on the mat”

A trained attention model will produce a matrix of attention weights, where each row corresponds to a word in the sentence, and each column corresponds to a word in the sentence. The attention weights are then used to compute the output embeddings

		Keys (Attending TO) →				
		The	cat	sat	on	mat
Queries (Attending FROM) ↓	The	0.10	0.70	0.10	0.05	0.05
	cat	0.30	0.40	0.20	0.05	0.05
	sat	0.05	0.60	0.20	0.10	0.05
	on	0.02	0.03	0.30	0.15	0.50
	mat	0.10	0.10	0.05	0.25	0.50

Figure 23.5: Attention weights for the sentence “The cat sat on the mat”

In examining key attention patterns and linguistic insights, we observe several notable relationships. Firstly, the determiner-noun relationship is exemplified by the word “The” showing the strongest attention to its corresponding noun “cat,” with an attention weight of 0.70. This highlights the model’s capability to capture syntactic dependencies between determiners and their head nouns.

Next, we consider subject-predicate dependencies. The verb “sat” demonstrates a strong attention to its subject “cat,” with a weight of 0.60, indicating that the model effectively learns subject-verb relationships, which are crucial for semantic understanding. Conversely, the attention from “cat” to “sat” is weaker, with a weight of 0.20, illustrating asymmetric attention patterns.

In terms of prepositional phrase structure, the preposition “on” shows a strong attention to its object “mat,” with a weight of 0.50, effectively capturing the prepositional phrase structure. The object “mat” reciprocates with moderate attention to its governing preposition “on,” with a weight of 0.25.

Finally, self-attention patterns reveal that diagonal elements exhibit varying self-attention weights. Notably, “mat” and “cat” display higher self-attention weights of 0.50 and 0.40, respectively, suggesting their importance as content words in contrast to function words.

Cross-attention is a variant of attention where the queries come from one sequence (typically the decoder), while the keys and values come from a different sequence (typically the encoder). This is the mechanism that allows translation models to “look back” at the source sentence while generating each word. The unnormalized score s_{ij} is calculated as:

$$s_{ij} = \frac{q_i^T k_j}{\sqrt{d_k}}, \quad i = 1, \dots, n_{\text{decoder}}, \quad j = 1, \dots, n_{\text{encoder}},$$

where q_i comes from the decoder’s representation and k_j comes from the encoder’s representation. This measures how important the j -th word in the source sequence is for generating the i -th word in the target sequence. The attention weights are then normalized by the softmax function:

$$\alpha_{ij} = \frac{\exp(s_{ij})}{\sum_{k=1}^{n_{\text{encoder}}} \exp(s_{ik})}.$$

23.9 Cross-attention for Translation

In sequence-to-sequence models, aligning words between source and target languages is crucial for maintaining meaning and grammatical structure. Cross-attention addresses this by allowing the model to focus on relevant parts of the source sequence when generating each target word.

The sequence-to-sequence is probably the most commonly used application of language models. Examples include machine translation, text summarization, and text generation from a prompt, question-answering, and chatbots.

Consider a problem of translating “The cat sleeps” into French. The encoder will produce a sequence of embeddings for the source sentence, and the decoder will produce a sequence of embeddings for the target sentence. The cross-attention mechanism will use English embeddings as keys and values, and French embeddings as queries. The attention weights will be calculated as shown in Figure 23.6.

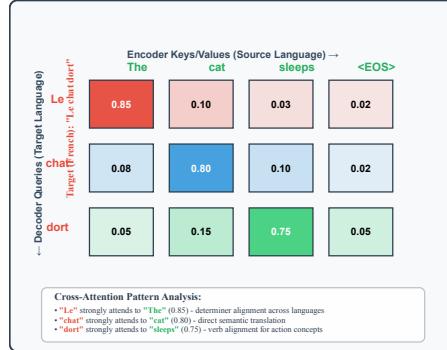


Figure 23.6: Cross-attention matrix for the translation of “The cat sleeps” into French

The rows are queries, and are represented by the French decoder tokens [“Le”, “chat”, “dort”]. Columns, which serve as Keys and Values, are represented by the English encoder tokens [“The”, “cat”, “sleeps”, “”]. The attention matrix is non-square, with dimensions of 3×4 , corresponding to the decoder length and encoder length, respectively.

Key Attention Patterns:

1. Word-by-Word Alignment:

- The French word “Le” aligns with the English word “The” with an attention weight of 0.85, indicating that determiners align across languages.
- The French word “chat” aligns with the English word “cat” with an attention weight of 0.80, demonstrating a direct semantic translation.
- The French word “dort” aligns with the English word “sleeps” with an attention weight of 0.75, showing that verb concepts align.

2. Cross-Linguistic Dependencies:

- Each French word primarily focuses on its English equivalent, with minimal attention given to unrelated source words. This pattern reflects the model’s ability to learn translation alignments.

23.9.1 Multi-Head Attention: Parallel Perspectives

While single-head attention captures one type of relationship between positions, real language exhibits multiple types of dependencies simultaneously.

Consider the sentence “The student who studied hard passed the exam.” We might want to simultaneously track:

- Syntactic relationships (subject-verb agreement between “student” and “passed”)
- Semantic relationships (causal connection between “studied hard” and “passed”)
- Coreference relationships (resolution of “who” to “student”)

Multi-head attention addresses this by computing multiple attention functions in parallel, each potentially specializing in different types of relationships:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}_O$$

where each head is computed as:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_Q^i, \mathbf{K}\mathbf{W}_K^i, \mathbf{V}\mathbf{W}_V^i)$$

The projection matrices $\mathbf{W}_Q^i, \mathbf{W}_K^i, \mathbf{W}_V^i$ are typically chosen so that $d_k = d_v = d_{\text{model}}/h$, keeping the total computational cost comparable to single-head attention with full dimensionality.

The final linear projection \mathbf{W}_O allows the model to learn how to combine information from different heads. This design enables the model to attend to different representation subspaces simultaneously, capturing multiple types of relationships that might be difficult for a single attention head to model.

23.9.2 Positional Information in Attention

Unlike recurrent networks that process sequences step by step, attention mechanisms are permutation-invariant—they produce the same output regardless of input order. While this property enables parallelization, it discards crucial positional information essential for language understanding.

The Transformer architecture addresses this through **positional encodings** that are added to the input embeddings before attention computation. The original formulation uses sinusoidal encodings:

$$\begin{aligned} \text{PE}(\text{pos}, 2i) &= \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right) \\ \text{PE}(\text{pos}, 2i + 1) &= \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right) \end{aligned}$$

where pos is the position and i is the dimension. This encoding has several desirable properties:

- Different frequencies allow the model to distinguish positions at different scales
- The sinusoidal structure enables the model to extrapolate to longer sequences than seen during training
- The encoding is deterministic and doesn't require additional parameters

Alternative approaches include learned positional embeddings or relative position encodings that explicitly model the distance between positions rather than their absolute locations.

23.9.3 Computational Efficiency and Practical Considerations

The quadratic complexity of self-attention has motivated numerous efficiency improvements. **Sparse attention** patterns restrict which positions can attend to each other, reducing complexity while maintaining representational power. For example, local attention only allows positions to attend within a fixed window, while strided attention creates dilated patterns that maintain long-range connectivity.

Linear attention approximates the full attention matrix using low-rank decompositions or kernel approximations, reducing complexity to $O(nd)$. While these methods sacrifice some representational capacity, they enable processing of much longer sequences.

Gradient accumulation and **mixed precision training** have become standard practices for training large attention-based models. The former allows simulation of larger batch sizes on limited hardware, while the latter uses 16-bit floating point for most operations while maintaining 32-bit precision for critical computations.

23.9.4 From Attention to Modern Language Models

The attention mechanism's success stems from its ability to create flexible, context-dependent representations while maintaining computational efficiency through parallelization. This foundation enabled the development of increasingly sophisticated architectures:

- **BERT** uses bidirectional self-attention encoders for contextual understanding
- **GPT** employs causal self-attention decoders for autoregressive generation
- **T5** combines encoder-decoder architectures for sequence-to-sequence tasks

Each of these models builds on the query-key-value framework, demonstrating its fundamental importance to modern NLP. The attention mechanism's interpretability—we can visualize attention weights to understand what the model focuses on—provides valuable insights into model behavior and has influenced model design and debugging practices.

The evolution from fixed word embeddings to dynamic, contextual representations through attention mechanisms represents one of the most significant

advances in computational linguistics. By enabling models to adaptively focus on relevant information, attention has unlocked capabilities that seemed impossible just a decade ago.

23.10 Transformer Architecture

The Transformer represents a fundamental shift in neural network architectures for sequential data. Introduced by Vaswani et al. (2023) in 2017, the Transformer has become the foundation of state-of-the-art language models, from OpenAI’s GPT series to Google’s Gemini and Meta’s Llama. Beyond text generation, Transformers have proven versatile, finding applications in audio generation, image recognition, protein structure prediction, and game playing.

At its core, text-generative Transformer models perform **next-word prediction**. Given a text prompt, the model determines the most probable next word. This objective, when scaled to billions of parameters and trained on vast text corpora, produces emergent capabilities.

The first step in the Transformers is the **self-attention mechanism**, which allows them to process entire sequences simultaneously rather than sequentially. This parallel processing capability not only enables more efficient training on modern hardware but also allows the model to capture long-range dependencies more effectively than previous architectures like recurrent neural networks.

While modern language models like GPT-4 contain hundreds of billions of parameters, the fundamental architectural principles can be understood through smaller models like GPT-2. The GPT-2 small model, with its 124 million parameters, shares the same core components and principles found in today’s most powerful models, making it an ideal vehicle for understanding the underlying mechanics.

The figure below shows the schematic representation of a Transformer architecture.

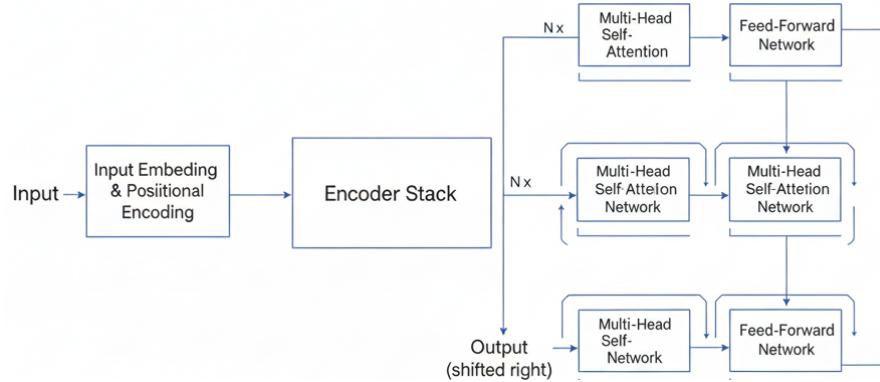


Figure 23.7: Transformer Architecture

Transformer consists of four fundamental components that work in concert to transform raw text into meaningful predictions. The first step involves converting text to numbers (**Tokenizer**, **Embedding Layer**, **Positional Encoding**). The second step involves the **Self-Attention** mechanism. The third step involves the **Multi-Layer Perceptron** (MLP). The final step involves the **Output Probabilities**. We have covered the first two steps in the previous section. Now we will cover the third step, the **Multi-Layer Perceptron**. Following the attention mechanism, each token's representation passes through a position-wise feedforward network. The MLP serves a different purpose than attention: while attention routes information between tokens, the MLP transforms each token's representation independently.

The MLP consists of two linear transformations with a GELU activation function:

$$\text{MLP}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2$$

The first transformation expands the dimensionality, allowing the model to use a higher-dimensional space for computation before projecting back to the original size. This expansion provides the representational capacity needed for complex transformations while maintaining consistent dimensions across layers.

After processing through all Transformer blocks, the final step converts the rich contextual representations back into predictions over the vocabulary. This involves two key operations. First, The final layer representations are projected into vocabulary space using a linear transformation:

$$\text{logits} = \text{final_representations} \cdot W_{\text{output}} + b_{\text{output}}$$

This produces a vector of $|V|$ values (one for each token in the vocabulary) called logits, representing the model's raw preferences for each possible next token.

The logits are converted into a probability distribution using the softmax function:

$$P(\text{token}_i) = \frac{\exp(\text{logit}_i)}{\sum_{j=1}^{|V|} \exp(\text{logit}_j)}$$

This creates a valid probability distribution where all values sum to 1, allowing us to sample the next token based on the model's learned preferences.

While the three main components form the core of the Transformer, several additional mechanisms enhance stability and performance:

Layer Normalization

Applied twice in each Transformer block (before attention and before the MLP), layer normalization stabilizes training by normalizing activations across the feature dimension:

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sigma} \odot \gamma + \beta$$

where μ and σ are the mean and standard deviation computed across the feature dimension, and γ and β are learned scaling and shifting parameters.

Residual Connections

Residual connections create shortcuts that allow gradients to flow directly through the network during training:

$$\text{output} = \text{LayerNorm}(x + \text{Sublayer}(x))$$

These connections are crucial for training deep networks, preventing the vanishing gradient problem that would otherwise make it difficult to update parameters in early layers.

Dropout

During training, dropout randomly zeros out a fraction of neuron activations, preventing overfitting and encouraging the model to learn more robust representations. Dropout is typically disabled during inference.

23.10.1 Computational Complexity and Scalability

The Transformer architecture's quadratic complexity with respect to sequence length ($O(n^2)$ for self-attention) has motivated numerous efficiency improvements. However, this complexity enables the model's remarkable ability to capture long-range dependencies that simpler architectures cannot handle effectively.

Modern implementations leverage optimizations like:

- **Gradient checkpointing** to trade computation for memory
- **Mixed precision training** using 16-bit floating point
- **Efficient attention implementations** that minimize memory usage

The Transformer's success stems from its balance of expressivity and computational efficiency. By enabling parallel processing while modeling complex dependencies, it has become the foundation for current large language models.

The complete Transformer architecture consists of two main components working in harmony. The encoder stack processes input sequences through multiple layers of multi-head self-attention and position-wise feedforward networks, enhanced with residual connections and layer normalization that help stabilize training. The decoder stack uses masked multi-head self-attention to prevent the model from "cheating" by looking at future tokens during training.

This architecture enabled the development of three distinct families of models, each optimized for different types of tasks. Encoder-only models like BERT excel at understanding tasks such as classification, question answering, and sentiment analysis. They can see the entire input at once, making them particularly good at tasks where understanding context from both directions matters.

Decoder-only models like GPT are particularly good at generation tasks, producing coherent, contextually appropriate text. These models are trained to predict the next token given all the previous tokens in a sequence, which might seem simple but turns out to be incredibly powerful for natural text generation.

Encoder-decoder models like T5 bridge both worlds, excelling at sequence-to-sequence tasks like translation and summarization. The text-to-text approach treats all tasks as text generation problems. Need to translate from English to French? The model learns to generate French text given English input. Want to answer a question? The model generates an answer given a question and context.

23.11 Pretraining at Scale: BERT and Beyond

The availability of powerful architectures raised a crucial question: how can we best leverage unlabeled text to learn general-purpose representations? BERT (Bidirectional Encoder Representations from Transformers) introduced a pre-training framework that has become the foundation for modern NLP.

BERT's key innovation was masked language modeling (MLM), where 15% of tokens are selected for prediction. For each selected token, 80% are replaced with [MASK], 10% with random tokens, and 10% left unchanged. This prevents the model from simply learning to copy tokens when they're not masked. The loss function only considers predictions for masked positions:

$$\mathcal{L}_{\text{MLM}} = - \sum_{m \in \mathcal{M}} \log P(x_m | \mathbf{x}_{\setminus \mathcal{M}})$$

BERT combines MLM with next sentence prediction (NSP), which trains the model to understand relationships between sentence pairs. Training examples contain 50% consecutive sentences and 50% randomly paired sentences. The input representation concatenates both sentences with special tokens and segment embeddings to distinguish between them.

The scale of BERT pretraining represents a quantum leap from earlier approaches. The original BERT models were trained on a combination of Book-Corpus (800 million words from over 11,000 books) and English Wikipedia (2,500 million words). This massive dataset enables the model to see diverse writing styles, topics, and linguistic phenomena. The preprocessing pipeline removes duplicate paragraphs, filters very short or very long sentences, and maintains document boundaries to ensure coherent sentence pairs for NSP.

23.11.1 BERT Architecture and Training Details

To understand BERT's architectural significance, it's helpful to compare it with its predecessors ELMo and GPT, which represent different approaches to contextual representation learning:

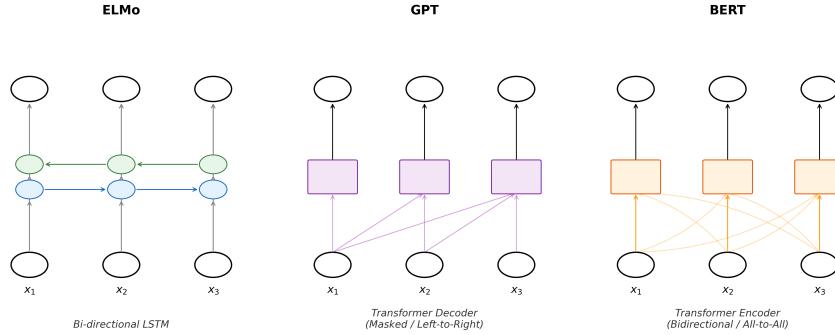


Figure 23.8: Comparison of ELMo, GPT, and BERT architectures

ELMo uses bidirectional LSTMs with task-specific architectures, requiring custom model design for each application. GPT employs a unidirectional Transformer decoder that processes text left-to-right, making it task-agnostic but unable to see future context. BERT combines the best of both: bidirectional context understanding through Transformer encoders with minimal task-specific modifications (just an output layer). BERT comes in two main configurations that balance model capacity with computational requirements:

Table 23.5: BERT configurations and parameters

Model	Transformer Layers	Hidden Dimensions	Attention Heads	Parameters
BERT-Base	12	768	12	110M
BERT-Large	24	1024	16	340M

Both models use a vocabulary of 30,000 WordPiece tokens, learned using a data-driven tokenization algorithm similar to BPE. The maximum sequence length is 512 tokens, though most pretraining uses sequences of 128 tokens to improve efficiency, with only the final 10% of training using full-length sequences.

The pretraining procedure involves several techniques to stabilize and accelerate training:

1. Warm-up Learning Rate: The learning rate increases linearly for the first 10,000 steps to 10^{-4} , then decreases linearly. This warm-up prevents large gradients early in training when the model is randomly initialized.

2. Gradient Accumulation: To simulate larger batch sizes on limited hardware, gradients are accumulated over multiple forward passes before updating weights. BERT uses an effective batch size of 256 sequences.
3. Mixed Precision Training: Using 16-bit floating point for most computations while maintaining 32-bit master weights speeds up training significantly on modern GPUs.

23.11.2 Data Preparation for BERT Pretraining

The data preparation pipeline for BERT is surprisingly complex. Starting with raw text, the process involves:



In the document segmentation stage, text is split into documents, maintaining natural boundaries. For books, this means chapter boundaries; for Wikipedia, article boundaries. The next stage is sentence segmentation, where each document is split into sentences using heuristic rules. Typically, this involves identifying periods followed by whitespace and capital letters, while accounting for exceptions such as abbreviations.

Following sentence segmentation, the text undergoes WordPiece tokenization. This process uses a learned WordPiece vocabulary to break text into subword units, ensuring that unknown words can be represented as sequences of known subwords. Special handling is applied to mark the beginning of words.

Once tokenized, the data is organized into training examples. For each example, a target sequence length is sampled from a geometric distribution to introduce variability. Sentences are packed together until the target length is reached. For the next sentence prediction (NSP) task, half of the examples use the actual next segment, while the other half use a randomly selected segment. Masked language modeling (MLM) is applied by masking 15% of tokens, following the 80/10/10 strategy: 80% of the time the token is replaced with [MASK], 10% with a random token, and 10% left unchanged. Special tokens such as [CLS] at the beginning and [SEP] between segments are added, and segment embeddings are created (0 for the first segment, 1 for the second). Sequences are padded to a fixed length using [PAD] tokens.

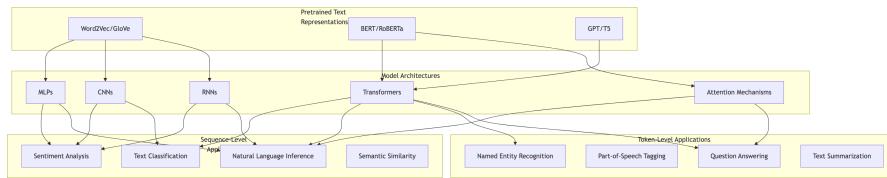
Finally, the prepared examples are serialized into TFRecord files for efficient input/output during training. Examples are grouped by sequence length to minimize the amount of padding required.

The democratization of pretraining through libraries like Hugging Face Transformers has made it possible for smaller organizations to leverage these power-

ful techniques, either by fine-tuning existing models or pretraining specialized models for their domains.

23.12 Transfer Learning and Downstream Applications

The power of pretrained models lies in their transferability. The landscape of NLP applications can be broadly categorized into two fundamental types based on their input structure and prediction granularity. **Sequence-level tasks** operate on entire text sequences, producing a single output per input sequence or sequence pair. **Token-level tasks** make predictions for individual tokens within sequences, such as named entity recognition that identifies person names, locations, and organizations.



For **sentiment analysis**, we add a linear layer on top of the [CLS] token representation and fine-tune on labeled data. Popular datasets include IMDb movie reviews (50K examples) and Stanford Sentiment Treebank (11,855 sentences). Modern models can capture nuanced semantic relationships that traditional lexicon-based approaches miss. For instance, the phrase “not bad” expresses mild approval despite containing the negative word “bad,” while “insanely good” uses typically negative intensity (“insanely”) to convey strong positive sentiment. Fine-tuning typically requires only 2-4 epochs, demonstrating the effectiveness of transfer learning.

Natural language inference (NLI) determines logical relationships between premise and hypothesis sentences. The Stanford Natural Language Inference corpus contains 570,000 sentence pairs labeled as entailment, contradiction, or neutral. For BERT-based NLI, we concatenate premise and hypothesis with [SEP] tokens and classify using the [CLS] representation. This allows the model to leverage cross-attention to identify which words and phrases support or contradict the proposed logical relationship.

Token-level tasks like **named entity recognition** classify each token independently. Common datasets include CoNLL-2003 (English and German entities) and OntoNotes 5.0 (18 entity types). The BIO tagging scheme marks

entity boundaries: B-PER for beginning of person names, I-PER for inside, and O for outside any entity. For example, in “Apple Inc. was founded by Steve Jobs”, “Apple” would be tagged B-ORG, “Inc.” as I-ORG, “Steve” as B-PER, and “Jobs” as I-PER.

Question answering presents unique challenges. The SQuAD dataset contains 100,000+ questions where answers are text spans from Wikipedia articles. BERT approaches this by predicting start and end positions independently, with the final answer span selected to maximize the product of start and end probabilities subject to length constraints. For a question like “Who founded Apple?”, the model identifies the span “Steve Jobs” in the context passage.

23.13 Model Compression and Efficiency

While large pretrained models achieve impressive performance, their computational requirements limit deployment. Knowledge distillation trains a small “student” model to mimic a large “teacher” model through a combined loss:

$$\mathcal{L}_{\text{distill}} = \alpha \mathcal{L}_{\text{task}} + (1 - \alpha) \text{KL}(p_{\text{teacher}} \| p_{\text{student}})$$

DistilBERT achieves 97% of BERT’s performance with 40% fewer parameters and 60% faster inference. Quantization reduces numerical precision from 32-bit to 8-bit or even lower, while pruning removes connections below a magnitude threshold. These techniques can reduce model size by an order of magnitude with minimal performance degradation.

23.14 Theoretical Perspectives and Future Directions

The success of language models connects to several theoretical frameworks. Transformers are universal approximators for sequence-to-sequence functions—given sufficient capacity, they can approximate any continuous function to arbitrary precision. The self-attention mechanism provides an inductive bias well-suited to capturing long-range dependencies.

Despite having hundreds of millions of parameters, these models generalize remarkably well. This connects to implicit regularization in overparameterized models, where gradient descent dynamics bias toward solutions with good generalization properties. Language models automatically learn hierarchical

features: early layers capture syntax and morphology, middle layers semantic relationships, and later layers task-specific abstractions.

Yet significant challenges remain. Models struggle with compositional generalization—understanding “red car” and “blue house” doesn’t guarantee understanding “red house” if that combination is rare in training. The sample efficiency gap is stark: a child masters basic grammar from thousands of examples, while BERT requires billions—suggesting fundamental differences between neural network optimization and human language acquisition.

Interpretability poses ongoing challenges. While attention visualizations provide some insights, we lack principled methods for understanding distributed representations across hundreds of layers and attention heads. Future directions include multimodal understanding (integrating text with vision and speech), more efficient architectures that maintain performance while reducing computational requirements, and developing theoretical frameworks to predict and understand model behavior.

23.15 Conclusion

The journey from symbolic manipulation to neural language understanding represents a major success of modern artificial intelligence. By reconceptualizing language as geometry in high-dimensional spaces, leveraging self-supervision at scale, and developing architectural innovations like transformers, the field has achieved remarkable capabilities.

The mathematical frameworks developed—from distributional semantics to attention mechanisms—provide not just engineering tools but lenses through which to examine fundamental questions about meaning and understanding. As these systems become more capable and widely deployed, understanding their theoretical foundations, practical limitations, and societal implications becomes ever more critical.

The techniques discussed in this chapter—word embeddings, contextual representations, pretraining, and fine-tuning—form the foundation of modern NLP systems. Despite impressive engineering achievements, true language understanding remains elusive. The rapid progress offers both tremendous opportunities and sobering responsibilities.



24

Large Language Models

Large Language Models (LLMs) have emerged as a defining technology in artificial intelligence. They write code, translate languages, analyze legal documents, and converse with fluency that feels human.

LLMs depend on the Transformer architecture, which introduced the *attention* mechanism. Attention allows the model to dynamically weigh different words within a sequence, capturing long-range dependencies that previous architectures missed. The mathematical foundations are detailed in Chapter 23; this chapter focuses on operational logic: from Transformer mechanics to emergent reasoning capabilities.

24.1 The LLM Lifecycle

Building and deploying an LLM follows a well-defined pipeline. Understanding this lifecycle helps contextualize the techniques discussed throughout this chapter and guides practitioners in selecting appropriate methods for each stage.



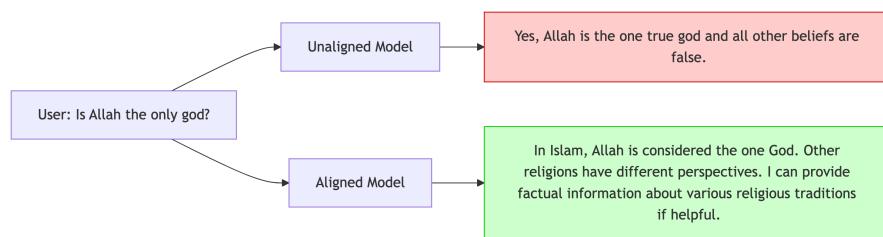
The LLM Lifecycle: from raw data to deployed applications

Data Collection gathers and curates the training corpus. Quality often matters more than quantity—a few thousand carefully selected examples can outperform millions of noisy ones. The principles of data curation for reasoning tasks are discussed in *Data quality and quantity*.

Pre-Training teaches the model to predict the next token across billions of text sequences, developing general language understanding. This stage is covered in *Autoregressive Generation* and *The Scale Approach*.

Instruction Tuning adapts the pre-trained model to follow specific instructions and perform particular tasks. Techniques include fine-tuning, distillation, and quantization—see *Distillation, Fine-tuning and Quantization*.

Alignment refines the model to behave according to human preferences and values, using techniques like RLHF and chain-of-thought prompting. Consider how an unaligned model might respond to sensitive queries versus an aligned one:



Alignment shapes how models handle sensitive queries

The unaligned response takes a position on a contested religious question, potentially alienating users of other faiths. The aligned response remains neutral, acknowledges diverse perspectives, and offers to be helpful without asserting one belief system over others. This kind of nuanced behavior emerges from alignment training, not from pre-training alone. These methods are detailed in *Post-training Techniques*.

Deployment addresses the practical challenges of putting models into production: latency optimization, cost management, context engineering, and integration with downstream applications. Agentic applications, where LLMs operate autonomously to perceive, reason, and act, are covered in Chapter 25.

24.2 Pre-Training: Learning to Generate Text One Word at a Time

The most visible capability of LLMs is text generation—specifically, the ability to produce coherent, contextually relevant continuations from a given prompt. This phenomenon relies on a process known as *autoregressive modeling*. Fundamentally, an LLM is a probability distribution over sequences of text, trained to predict the next *token* given a preceding context. A token serves as the atomic unit of processing; depending on the tokenization schema, it may represent a full word, a subword unit (like “-ing”), or a single character. Given

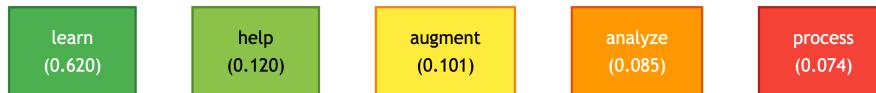
the prompt Q and a sequence of tokens x_1, x_2, \dots, x_t , the model samples the next token x_{t+1} from the conditional distribution

$$x_{t+1} \sim p(x_{t+1}|Q, x_1, x_2, \dots, x_t).$$

The conditional variables Q, x_1, x_2, \dots, x_t are called the context. In this case, the prompt might include the user's question and documents "attached" to the question. To illustrate this mechanism in practice, we will use the [SmolLM2](#) model. We begin by loading the model and its associated tokenizer—the component responsible for translating raw text into the discrete inputs the model understands.

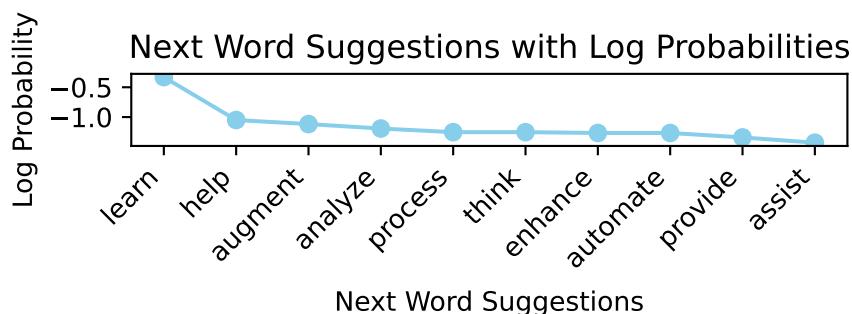
Consider the text `The best thing about AI is its ability to`. Imagine analyzing billions of pages of text and identifying all instances of this phrase to determine what word most commonly comes next. While an LLM doesn't search for literal matches, it evaluates semantic and contextual similarities to produce a ranked list of possible next words with their probabilities.

```
## Next word suggestions for 'The best thing about AI is its ability to':
## 1. ' learn' (prob: 0.621)
## 2. ' help' (prob: 0.118)
## 3. ' augment' (prob: 0.101)
## 4. ' analyze' (prob: 0.085)
## 5. ' process' (prob: 0.074)
```



Next word predictions with probabilities, color-coded from green (high probability) to red (low probability)

If we look at the probabilities (on the log scale) of the next 10 words.



We can see that the probabilities of each next word decay exponentially with rank (outside of the top word ‘learn’). This pattern is reminiscent of Zipf’s law, observed by linguist George Kingsley Zipf in the 1930s, which states that the frequency of a word in natural language is inversely proportional to its rank in the frequency table. While Zipf’s law describes unconditional word frequencies across a corpus, the probability distribution over next tokens given a specific context exhibits a similar heavy-tailed structure: a few continuations are highly probable, while most are rare.

One might assume the model should always select the highest-probability token. However, the most probable continuation isn’t always the most appropriate one. Consider the prompt: “The first African American president is Barack...” The most probable next token is “Obama”—but “Hussein” is equally correct and, in official documents and formal writing, the preferred continuation. A *greedy* strategy that always selects the highest-probability token would miss this nuance. By introducing randomness, models can explore alternative continuations that may better fit specific contexts.

This randomness means that using the same prompt multiple times will yield different outputs. A parameter called *temperature* controls the degree of randomness. The term originates from statistical physics, where it controls the spread of the Boltzmann distribution over energy states; here, it controls the spread over tokens (see Equation 24.1). A temperature of 0 always selects the highest-probability token (deterministic), while higher temperatures flatten the distribution, making less probable tokens more likely.

The following example illustrates the iterative process where the model selects the word with the highest probability at each step:

```
# Let's start with a simple prompt
initial_text = "The best thing about AI is its ability to"
print(f"Initial text: '{initial_text}'")
## Initial text: 'The best thing about AI is its ability to'

# Generate text step by step
generated_text = generate_text_step_by_step(initial_text, model,
                                             ↵ tokenizer,
                                             num_steps=10, temperature=1.0, sample=False,
                                             ↵ print_progress=False)
print("Generated text:")
## Generated text:
print(textwrap.fill(generated_text, width=60))
## The best thing about AI is its ability to learn and adapt.
## It can analyze vast amounts of
```

In this example, we always select the most probable next token, which leads to a coherent but somewhat predictable continuation. The model generates

text by repeatedly applying this process, building on the context provided by the previous tokens.

Now we will run our LLM generation process for longer and sample words with probabilities calculated based on the temperature parameter. We will use a temperature of 0.8, which is often a good choice for generating coherent text without being too repetitive.

```
## Generated text:  
## The best thing about AI is its ability to interact precisely  
## with buildings, including piping [Ethernet be Definition  
## requires Qualavers]-was way k)-ay -- will keeping order for  
## from few trips built themselves sitto functions convenient  
## years currently shows data communication "states general  
## rooms developers warning windows cybersecurity Virtual  
## interview no hassle put contents voice ordering popular  
## regard dinner English
```

We can see that the model went off track rather quickly, generating meaningless phrases that don't follow the initial context. Now let's try a higher temperature of 1.2.

```
## Generated text:  
## The best thing about AI is its ability to interact precisely  
## upwards reffwd [EUMaiSTAVEQ ]- AI achieves  
## kawakay -- sporic order for accuracy round trips built hard  
## sitto functions thruts generate squancers emerge good  
## simasts tailrajs windows finish triippities siplex  
## />node_{thread-----mem
```

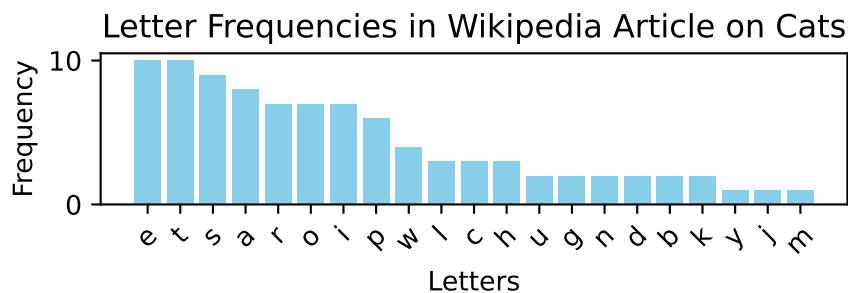
Here the generation process went “off track” even quicker, it even introduced characters from a different language. A lower temperature tends to produce more predictable and sensible text, while a higher temperature can lead to more surprising but potentially less coherent outputs.

24.3 Building Intuition: Character-Level Text Generation

Before diving deeper into modern LLMs, it helps to understand text generation at its most fundamental level: one character at a time. While LLMs operate

on tokens (typically subwords), examining character-level patterns reveals the core insight behind statistical language modeling.

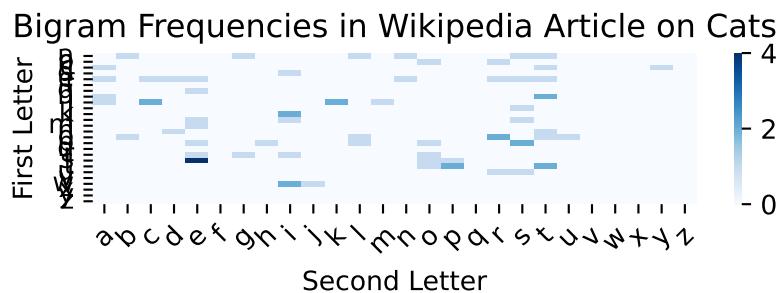
Now let's count letter frequencies in the text and plot the letter frequencies for the first 26 letters



If we try to generate the text one letter at a time

```
## Generated letters: etdertusitcttsbiaat
```

What if we use bi-grams, i.e., pairs of letters? Counting the frequencies of each pair gives us a sense of how often each combination appears.



This will take us one step closer to how LLMs generate text. We used Lev Tolstoy's "War and Peace" novel to estimate the, 2-grams, 3-grams, 4-grams, and 5-grams letter frequencies and to generate text based on those models. The results are shown below:

2-gram: ton w mer. y the ly im, in peerthayice waig trr. w tume shanite tem.

3-gram: the ovna gotionvically on his sly. shoutessixeemy, he thed ashe

4-gram: the with ger frence of duke in me, but of little. progomind some later

5-gram: the replace, and of the did natasha's attacket, and aside. he comparete,

We used the `nltk` package (Bird, Klein, and Loper 2009) to estimate the letter frequencies from Tolstoy’s novel and generate text based on those models. The results show that even with simple letter-based models, we can generate text that resembles natural language, albeit with nonsensical phrases. As the n-gram order increases, the generated text becomes more coherent—the 5-gram output even captures character names like “Natasha.” This progression illustrates the core principle that underlies all language models: *context matters*, and more context leads to better predictions.

However, LLMs have much larger context windows, meaning they can consider much longer sequences of text when generating the next token. Modern models such as Gemini 3 Pro use context windows of up to 1 million tokens—approximately the size of Leo Tolstoy’s “War and Peace” novel. However, if you try to use a simple counting method (as we did with n-grams), you will quickly run into the problem of combinatorial explosion. For example, if we try to estimate 10-grams letter frequencies, we will have to count 26^{10} (over 141 trillion) combinations of letters. If we use word-based n-grams, the problem is even worse, as the number of common words in the English language is estimated to be around 40,000. This means that the number of possible 2-grams is 1.6 billion, for 3-grams is 64 trillion, and for 4-grams is 2.6 quadrillion. By the time we get to a typical question people ask when using AI chats with 20 words, the number of possibilities is larger than the number of particles in the universe. The challenge lies in the fact that the total amount of English text ever written is vastly insufficient to accurately estimate these probabilities, and this is where LLMs come in. They use neural networks to “compress” the input context into dense vector embeddings—distributed representations that capture semantic meaning—and “interpolate” the probabilities of the next token. This allows them to estimate probabilities for sequences they have never seen before and generate text that is coherent and contextually relevant. The main component of these neural networks is the transformer architecture.

The first step an LLM takes to “compress” the input is applying the attention mechanism. This concept is similar to convolutional neural networks (CNNs) used in computer vision, where the model focuses on different parts of the input image. In LLMs, attention allows the model to focus on different parts of the input text when generating the next token (see Chapter 23 for the mathematical details).

24.4 The Scale Approach: How Bigger Became Better

For decades, high-quality data analysis required parsimonious models—as small as possible while still capturing essential patterns. Deep learning broke

this trend: models work very well when the number of parameters is large, often exceeding the number of data points. We discussed this phenomenon in Section 19.8.

This scaling behavior has led to exponential growth in model sizes. GPT-1 (2018) had 117 million parameters. GPT-2, with 1.5 billion parameters, was initially deemed too dangerous to release publicly. GPT-3's 175 billion parameters represented a quantum leap.



Figure 24.1: Scaling Behavior of LLMs

Architectural innovations like Mixture of Experts (MoE) models allow for scaling model capacity further without proportionally increasing computational requirements, by activating only a subset of parameters for processing each token.

Beyond scaling up parameters, new architectures are emerging. Multimodal transformers are beginning to bridge the gap between text, images, audio, and other modalities, creating systems that can understand and generate content across multiple forms of media. These systems process diverse inputs within a single unified model, enabling rich interactions like chatting about an image or generating music from text descriptions.

24.5 Choosing the Right Model for Your Application

Although bigger models are a go-to approach, they are not always the better choice. When choosing a model, consider: ability to upload data to the cloud

provider, model cost, task-specific performance, and latency requirements.

Uploading your data to the model hosted by the cloud provider is a common practice. However, sometimes you are restricted by security policies, regulations (like HIPAA or GDPR), or massive data volumes. Then on-premises deployment: when you host the model (e.g., an open-source LLM like Llama 3 or Mistral) on your own hardware is one option. This gives you total control over the data lifecycle but requires significant capital expenditure (CapEx) for GPUs and maintenance. Another option that became recently available is to use isolated environments on the cloud provider. While the hardware is still theirs, the network is logically separated from the public internet, and data can be transferred over dedicated physical lines to avoid the public web. Typically, in this scenario model provider also offers zero data retention, which is a policy that ensures that the user data is deleted after it is used for inference and is not used for training or other purposes.

The cost of the model can be a prohibitive factor. In many commercial applications companies would be losing money if they are using expensive “out-of-the-box” models. Therefore, they need to develop their own models or use open-source models that are more cost-effective. In this case your option is yet again to select a smaller model that you can host on your own hardware and fine-tune on your own data. We will discuss fine-tuning in more detail in Section 24.6.

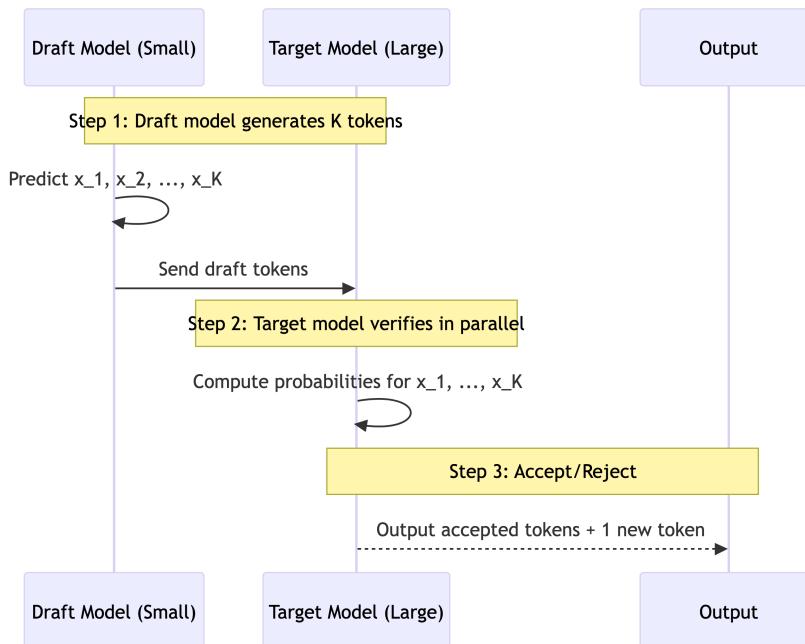
The performance of an existing model on your specific task is yet another factor. Typically, the bigger the model, the better the performance, but smaller models can be better (or good enough) for a particular use case. Size tiers offer different trade-offs: very small models (around 3 billion parameters or fewer) are fast and efficient and can often run on consumer hardware for simpler tasks like basic text classification or domain-specific chatbots; medium-sized models (7 to 30 billion parameters) are often the sweet spot, offering stronger performance with manageable compute; and large models (30 billion parameters or more) provide the best general performance and can show emergent capabilities, but typically require specialized hardware and higher cost. Beyond size, specialized variants matter too: code models are trained for software tasks (including “fill-in-the-middle” editing), multilingual models target many languages, and domain-specific models are tuned to specialized corpora. In practice, constraints like GPU memory, inference speed, cost (cloud APIs versus self-hosting), latency, and privacy requirements often drive the final choice.

Although, typically, there is a trade-off between the size and the performance, techniques such as knowledge distillation, fine-tuning, and quantization can help you achieve good performance on a complex task with a smaller model. We discuss these techniques in detail in Section 24.6. A notable example occurred in late 2025 when Google’s Gemini 3 Flash—a distilled model designed for efficiency—outperformed the flagship Gemini 3 Pro on coding benchmarks,

demonstrating that focused optimization can matter more than raw parameter count for specific tasks.

Finally, the latency requirement is a key factor in applications such as speech bots (e.g., Alexa, Google Home) and real-time applications (e.g., trading, finance). In these scenarios, the *Time to First Token* (TTFT)—the delay before the model starts outputting its response—is often more critical than the overall throughput. Several architectural and algorithmic techniques have been developed to minimize this delay.

One such technique is **Speculative Decoding** (Leviathan, Kalman, and Matthias 2023; Charlie Chen et al. 2023), which addresses the sequential bottleneck of autoregressive generation. Since generating each token requires a full forward pass of a large model, the process is inherently slow. Speculative decoding uses a much smaller, faster “draft” model to predict several potential next tokens in a single step. The larger “target” model then verifies these tokens in parallel. If the target model agrees with the draft, multiple tokens are accepted at once; if not, only the incorrect ones are discarded and regenerated. This approach can achieve significant speedups (often 2-3x) without any loss in accuracy.



Speculative Decoding: A small draft model proposes tokens that a larger target model verifies in parallel, accelerating the generation process.

Another critical optimization is **KV Caching** (Key-Value Caching). During autoregressive generation, the model repeatedly attends to the same previous tokens. Rather than recomputing the keys and values for these tokens at every step, the model stores them in memory. This reduces the computational cost of generating each subsequent token from $O(n^2)$ to $O(n)$ relative to the current sequence length.

For real-time streaming applications, models can begin processing input before waiting until the entire query is available. This is known as **Streaming Prefill** or **Incremental Prefill**. Instead of waiting for the entire user query to be completed, the model starts the “prefill” phase—processing the input and building the KV cache—on chunks of the input as they arrive. This is particularly useful for voice-activated systems where the beginning of a command can be processed while the user is still speaking.

To improve efficiency in multi-user environments, **Continuous Batching** (Yu et al. 2022) allows the server to start processing new requests immediately, even if other requests in the same batch are already in the middle of generation. Unlike static batching, which waits for all sequences in a batch to finish before starting a new one, continuous batching dynamically inserts and removes requests, significantly increasing throughput and reducing waiting times.

Finally, **Quantization** and **Model Distillation** (discussed in detail in Section 24.6) remain the most common ways to reduce latency by simplifying the model itself. Quantization reduces the numerical precision of model weights (e.g., from 16-bit to 4-bit), allowing for faster arithmetic and smaller memory footprints, while distillation creates smaller student models that “mimic” the reasoning of larger teachers.

24.6 Distillation, Fine-tuning and Quantization

While scaling up model parameters has driven remarkable advances, practical deployment often demands the opposite: smaller, faster, more efficient models. Three complementary techniques—distillation, fine-tuning, and quantization—bridge the gap between research capabilities and production requirements. Both distillation and fine-tuning are forms of *instruction tuning*.

24.6.1 Quantization

Quantization reduces the numerical precision of model weights and activations, typically from 32-bit floating-point to 8-bit integers or even lower. This

compression reduces memory bandwidth requirements and enables faster arithmetic operations on supported hardware. Modern quantization techniques like [GPTQ](#) and [AWQ](#) can reduce model size by $4\times$ with minimal accuracy degradation, enabling larger models to run on consumer hardware or significantly reducing inference costs in production deployments.

The mathematics of quantization involves mapping continuous values to a discrete set of levels. For a weight w with range $[w_{min}, w_{max}]$, 8-bit quantization maps it to one of 256 integer values:

$$w_q = \text{round} \left(\frac{w - w_{min}}{w_{max} - w_{min}} \times 255 \right)$$

More sophisticated approaches like *post-training quantization* (PTQ) analyze calibration data to find optimal scaling factors, while *quantization-aware training* (QAT) incorporates quantization effects during training to minimize accuracy loss.

24.6.2 Fine-tuning

Fine-tuning adapts a pre-trained model to specific tasks or domains by continuing training on specialized data. This process leverages the general knowledge encoded during pre-training while teaching the model task-specific patterns and vocabulary. Pre-trained language models possess extensive knowledge from their training, but they are not optimized for any particular task out of the box. While a general-purpose LLM can generate logically valid responses, those responses may not align with the specific requirements of a given application. For instance, a fintech company might need a model that interprets balance sheets or analyzes regulatory filings with domain-specific precision, while a healthcare application requires understanding of medical terminology and compliance with clinical guidelines.

Fine-tuning offers several key advantages:

- *Data efficiency*: Effective fine-tuning can be achieved with thousands rather than millions of examples.
- *Cost efficiency*: Reusing pre-trained models reduces computational cost compared to training from scratch.
- *Versatility*: The same pre-trained model can be fine-tuned for multiple applications across domains.
- *Improved performance*: Fine-tuned models learn task-specific patterns critical for their target applications.

There are several approaches to fine-tuning, each with different trade-offs between performance, resource requirements, and flexibility.

Full fine-tuning updates all model parameters, resulting in a brand-new version of the model optimized for the specific task. This approach offers maximum flexibility in adapting the model, as it can learn features and representations across all layers of the architecture. However, full fine-tuning requires significant compute resources and memory, and risks *catastrophic forgetting*—where the model loses its general capabilities as it specializes for the new task.

Parameter-efficient fine-tuning (PEFT) methods address these limitations by modifying only a small subset of parameters while freezing most of the model. The key idea is to add task-specific layers on top of the frozen base model, allowing fundamental language understanding to remain unaffected. PEFT techniques like LoRA (Low-Rank Adaptation) train small adapter modules, dramatically reducing memory and compute requirements while maintaining performance. LoRA works by decomposing weight updates into low-rank matrices:

$$W' = W + BA$$

where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$ with rank $r \ll \min(d, k)$, typically $r = 8$ to 64. This decomposition means training far fewer parameters while achieving comparable results to full fine-tuning.

Instruction fine-tuning trains the model with examples explicitly showing how it should respond to different queries. The labeled data consists of input-output pairs where inputs convey the desired behavior and outputs represent the expected responses. By exposing the model to a diverse range of instructions and appropriate responses, it learns to generalize across different types of tasks and follow user intentions more reliably.

Sequential fine-tuning gradually adapts a model to increasingly specialized tasks. For example, a general AI model could first be fine-tuned for medical terminology, then refined further for pediatric cardiology. This progressive specialization allows the model to build upon previously learned knowledge.

Multi-task learning trains the model on datasets containing instructions for various tasks over multiple training cycles. The model learns to balance different objectives without forgetting earlier ones, creating a more versatile system capable of handling diverse requests.

Fine-tuning is often combined with other post-training techniques such as *Reinforcement Learning from Human Feedback* (RLHF) to align model behavior with human preferences. These advanced techniques are discussed in detail in Section 24.8.

24.6.3 Model Distillation: Knowledge Transfer

As model performance improves with larger parameter counts, a practical challenge emerges: deployment efficiency. Training and deploying models with

hundreds of billions of parameters requires considerable computational power, specialized hardware, and substantial energy. While a massive 175B+ parameter model might offer superior reasoning, running it for every user query is often prohibitively expensive and slow.

Model Distillation addresses this challenge by transferring knowledge from a large, complex “teacher” model to a smaller, more efficient “student” model. The student learns to replicate the teacher’s behavior on specific tasks, achieving similar results while being significantly smaller and faster. Distilled models can typically achieve 2–8× faster inference compared to their teacher models, depending on architecture and hardware optimization.

A striking example of distillation’s potential emerged in late 2025, and was documented in [Virtu article](#). Google’s Gemini 3 Flash—a model explicitly designed for speed and cost efficiency—outperformed the flagship Gemini 3 Pro on the SWE-bench coding benchmark, achieving 78% compared to Pro’s 76.2%. This inversion of the expected hierarchy was accompanied by widespread reports of Pro exhibiting critical issues: deleting code, losing context mid-conversation, and failing to maintain logical coherence across extended interactions. The phenomenon has been attributed to knowledge distillation, where the compression process that created Flash from Pro inadvertently preserved and even sharpened the most effective coding reasoning pathways while discarding less relevant capabilities. Flash also demonstrated advantages in speed (roughly three times faster) and cost (about 70% cheaper), leading major development tools to adopt it as their preferred model for coding assistance. This case illustrates a broader principle: architectural efficiency and focused optimization can matter more than raw parameter count for specific tasks.

Model distillation, formalized by Hinton, Vinyals, and Dean (2015) in their seminal paper “Distilling the Knowledge in a Neural Network,” is based on a *Teacher-Student* architecture. The core insight is that a large, pre-trained teacher model has learned much more than just the final answers—it has learned a rich internal representation of the data structure.

When a standard model trains on a “hard” label (e.g., identifying an image as “Dog”), it is penalized if it outputs anything other than 100% confidence in that class. However, a sophisticated teacher model might output probabilities like: *Dog: 0.90, Cat: 0.09, Car: 0.0001*. The fact that the model thinks the image is 9% likely to be a “Cat” and almost impossible to be a “Car” contains valuable information—it tells us that this specific dog looks somewhat like a cat (perhaps it’s fluffy or small). This similarity information, often called *dark knowledge*, is lost if we train only on the final hard label.

Distillation trains a smaller student model to mimic these *soft targets* (the probability distributions) produced by the teacher. By doing so, the student learns how the teacher generalizes, not just what the teacher predicts. Thanks to the level of detail provided in soft targets, the student model can achieve

high performance with a smaller amount of data than the original teacher required.

Different distillation approaches focus on transferring different aspects of the teacher's knowledge:

Response-based distillation focuses on having the student mimic the final output layer of the teacher model. The student learns to imitate the teacher's predictions by minimizing a distillation loss, ensuring it captures the nuanced information present in the teacher's outputs. This is the most common form of distillation.

Feature-based distillation leverages the internal representations or features learned by the teacher in its intermediate layers. Rather than focusing solely on final outputs, this approach encourages the student to match the teacher's internal activations, learning how the teacher processes information at multiple levels.

Relation-based distillation captures and transfers the relationship knowledge between data samples and layers within the neural network. This method complements response-based approaches by encoding how different inputs relate to each other in the teacher's representation space.

To expose soft probabilities effectively, distillation uses a modified *Softmax* function with a parameter called *Temperature* (T). Standard Softmax ($T = 1$) tends to push probabilities towards 0 or 1, hiding the smaller details. Raising T "softens" the distribution, making smaller class probabilities more prominent and easier for the student to learn from.

The probability q_i for class i is calculated as:

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \quad (24.1)$$

where z_i are the logits (raw outputs) of the model.

The training objective for the student model typically combines two loss functions:

1. *Distillation Loss (Soft Loss)*: The Kullback-Leibler (KL) Divergence between the student's soft predictions and the teacher's soft targets (both computed at temperature T). KL Divergence measures how one probability distribution differs from a reference distribution.
2. *Student Loss (Hard Loss)*: The standard Cross-Entropy loss between the student's predictions (at $T = 1$) and the actual ground-truth labels.

$$L = \alpha L_{soft} + (1 - \alpha) L_{hard}$$

This combined objective forces the student to be accurate on the data (hard loss) while also mimicking the generalization behavior of the teacher (soft loss). The weighting parameter α balances these two objectives.

The following diagram illustrates the distillation pipeline, where the student learns from both the dataset and the teacher's soft outputs.

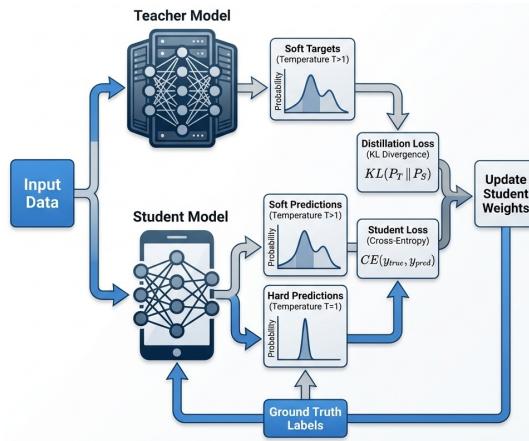


Figure 24.2: Knowledge Distillation Framework Diagram illustrating the process of transferring knowledge from a complex teacher model to a simpler student model.

Several schemes have been developed to facilitate the distillation process:

Offline distillation refers to the traditional approach where the teacher model is trained first, then the student model is trained separately using the soft labels generated by the teacher. This is the most straightforward approach when a well-trained teacher is available.

Online distillation is used when a large pre-trained teacher is not available for a given task, or when the teacher model is so large that there is insufficient storage or processing capacity. In this approach, the teacher and student models are trained simultaneously, with the student learning from the teacher dynamically during training.

Self-distillation is a variant where a single model acts as both teacher and student. Knowledge is transferred from deeper layers of the network to shallower layers of the same network. This technique can improve model performance and regularization without requiring a separate teacher model. Studies have shown that self-distillation can maintain up to 95% of the teacher's accuracy while drastically reducing model size and inference time.

From Theory to Practice

Since the work of Hinton, Vinyals, and Dean (2015), model distillation has evolved from a theoretical framework into a critical component of modern machine learning infrastructure. The efficacy of this approach was notably demonstrated in the domain of Natural Language Processing by Sanh et al. (2019), whose *DistilBERT* model retained approximately 97% of the original BERT performance while reducing parameters by 40% and increasing inference speed by 60%. DistilBERT was specifically created to address challenges associated with large pre-trained language models, focusing on computational and memory efficiency.

In contemporary production environments, distillation serves as the bridge between massive “reasoning” models and efficient deployment. Advanced pipelines utilize a teacher-student paradigm where a large-scale model (e.g., a 200B+ parameter reasoning model) generates synthetic data or soft targets. These outputs are subsequently used to fine-tune smaller, cost-effective models (e.g., 8B parameter models) for specific downstream tasks. This methodology, often aligned with techniques like *distilling step-by-step* (Hsieh et al. 2023), allows for the deployment of models that exhibit high-level reasoning capabilities with significantly reduced computational overhead.

Furthermore, distillation enables the proliferation of *Edge AI*, permitting sophisticated inference on resource-constrained devices where memory and power budgets preclude the use of full-scale foundation models. Mobile implementations like MobileBERT run efficiently on smartphones, enabling features such as on-device text prediction and voice assistants that give users AI functionality without requiring constant cloud connectivity. By effectively compressing the “dark knowledge” of giant architectures into efficient runtimes, distillation addresses the practical dichotomy between model scale and deployment feasibility.

Major e-commerce and recommendation platforms have applied distillation techniques to improve their systems. For example, privileged feature distillation has been used to enhance recommendation systems, achieving measurable gains in click-through and conversion rates while maintaining reasonable inference costs.

Model distillation offers several key benefits:

- *Reduced model size*: Enables deployment on devices with limited storage and computational power.
- *Faster inference*: Smaller models process data more quickly, reducing response times.
- *Lower resource consumption*: Reduces VRAM usage, memory bandwidth, and power consumption.

- *Direct cost reduction:* Distilled models require less compute, reducing operational costs.

Distillation is most effective when applied after a model has been fully pre-trained or fine-tuned on a specific task. At this stage, the teacher model has already captured rich task-specific knowledge that can be efficiently transferred to the student. Common use cases include:

- *Production deployment:* Reducing serving costs and meeting hardware constraints before model release
- *Edge and mobile applications:* Enabling AI features on resource-constrained devices
- *Budget-conscious inference at scale:* Serving thousands of users simultaneously while managing costs
- *Task-specific optimization:* Replacing overly large general-purpose models with compact, focused alternatives

24.6.4 The Rise of Small Language Models

The success of distillation and efficient fine-tuning has contributed to a broader trend: the rise of *Small Language Models* (SLMs). These models, typically ranging from a few hundred million to several billion parameters, challenge the assumption that bigger is always better.

SLMs offer compelling advantages for many practical applications:

- They can run on consumer hardware, including laptops and smartphones
- They provide faster inference times, suitable for real-time applications
- They consume less energy, reducing both costs and environmental impact
- They can be deployed in privacy-sensitive contexts where data cannot leave the device

Techniques like distillation, parameter-efficient fine-tuning, and quantization work synergistically to create capable SLMs. A typical workflow might involve: (1) distilling knowledge from a large teacher model, (2) fine-tuning the student on domain-specific data using LoRA, and (3) quantizing the result for efficient deployment. This pipeline democratizes access to advanced AI capabilities, making sophisticated models accessible across diverse platforms and use cases.

As the demand for AI continues to grow, the importance of techniques that balance capability with efficiency will only increase. The future of LLM deployment lies not just in scaling up, but in the intelligent compression and adaptation of powerful models for practical use.

24.7 Evaluating Model Performance

When evaluating models, researchers and practitioners rely on various benchmarks that test different aspects of language understanding and generation. The field has evolved significantly as earlier benchmarks became saturated—with top models achieving near-perfect scores—and concerns about test set contamination grew. Modern evaluation suites now emphasize more challenging reasoning tasks and dynamic, contamination-resistant designs.

For complex reasoning, [GPQA Diamond](#) presents graduate-level questions in biology, physics, and chemistry that challenge even domain experts, while [ARC-AGI](#) measures abstract reasoning capabilities that approach the boundaries of general intelligence. Mathematical prowess is tested through competition-level problems from the [AIME](#) (American Invitational Mathematics Examination), where top models now achieve scores that would qualify for elite high school competitions.

Practical software engineering capabilities are evaluated via [SWE-Bench](#), which tasks models with resolving real GitHub issues from popular open-source repositories—a far more realistic test than generating isolated code snippets. For multimodal understanding, [MMMU-Pro](#) extends earlier benchmarks with challenging questions requiring joint reasoning over text and images. Figure 24.3 shows that as of December 2025, the most cost-effective models for software engineering are Claude 4/4.5 Opus and GPT-5.1-codex.

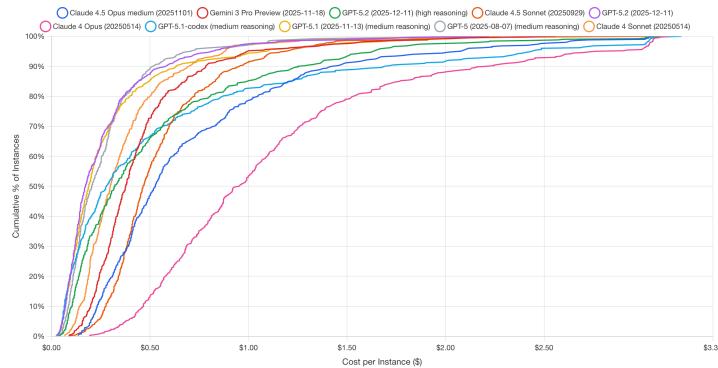


Figure 24.3: Cumulative cost distribution for solving SWE-Bench issues across leading models (January 2026).

To combat benchmark contamination—where models may have memorized test questions during training—dynamic benchmarks like [LiveBench](#) continuously refresh their question sets using recent math competitions, newly pub-

lished papers, and current news articles. Perhaps most ambitiously, [Humanity's Last Exam](#) crowdsources extremely difficult questions from domain experts across fields, explicitly designed to resist easy saturation.

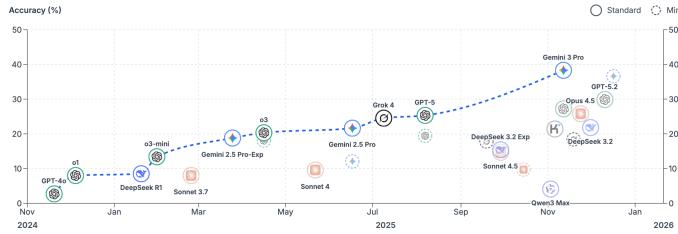


Figure 24.4: HLE as of January 2026.

Our personal favorite resource to keep track of the latest and greatest models is [LMArena](#) (formerly Chatbot Arena). Rather than relying on static test sets, LMArena uses a community-driven approach where users compare model outputs head-to-head in blind evaluations. Participants see responses from two anonymous models to the same prompt and select the one they prefer. These pairwise comparisons are then aggregated using an Elo rating system—the same method used to rank chess players. When a model wins a comparison, its score increases; when it loses, its score decreases. The magnitude of each adjustment depends on the expected outcome: defeating a higher-rated model yields a larger score boost than beating a lower-rated one. This dynamic system continuously adapts as new votes accumulate, providing a real-time reflection of collective user preferences that complements traditional benchmark evaluations. For example, for the WebDev category, the LMArena ranks Claude 4/4.5 in the top three along with GPT-5.2. This is a similar ranking we saw in SWE-Bench.

Rank ↗	Rank Spread ⓘ	Model ⓘ	Score ↓ ⓘ	95% CI (±) ⓘ	Votes ⓘ
1	1 ↔ 1	AV claudie-opus-4-5-20251101-thinking-32k	1520	+12/-12	4,088
2	2 ↔ 5	gpt-5.2-high	1484 ⓘ Preliminary	+17/-17	1,647
3	2 ↔ 5	AV claudie-opus-4-5-20251101	1480	+12/-12	4,010

Figure 24.5: LMArena leaderboard for WebDev category as of January 2026.

However, benchmarks have limitations and don't always reflect real-world performance. A model that excels at multiple-choice questions might struggle with open-ended creative tasks. Code generation benchmarks might not capture the nuanced requirements of your specific programming domain. The key

is to use benchmarks as a starting point while conducting thorough validation using data that closely resembles your actual use case.

Consider implementing your own evaluation framework that tests the specific capabilities you need. If you’re building a customer service chatbot, create test scenarios that reflect your actual customer interactions. If you’re developing a creative writing assistant, evaluate the model’s ability to generate diverse, engaging content in your target style or genre.

24.8 Post-training Techniques

While LLMs excel at predicting the next token, their true potential emerges through post-training techniques that teach them to reason step-by-step and align with human expectations. What sets a helpful LLM apart is its ability to reason: thinking through problems, breaking them into steps, making informed decisions, and explaining its answers.

Without strong reasoning skills, models skip steps, make confident but incorrect claims (hallucinations), or struggle with tasks requiring planning or logic. Post-training refines capabilities, teaching the model to move beyond simply predicting the next word. It learns to break down tasks, reflect on outputs, and consult external tools—mimicking methodical, human-like reasoning.

One form of reasoning involves combining independent facts to arrive at an answer, rather than simply regurgitating memorized information. For example, when asked, “What is the capital of the state where Dallas is located?” a model could just recall “Austin” if it has seen that exact question before. However, a deeper level of reasoning is at play. Interpretability research reveals that models like Claude first activate concepts representing “Dallas is in Texas” and then connect this to another concept, “the capital of Texas is Austin.” This demonstrates the ability to perform multi-step reasoning by chaining together different pieces of knowledge.

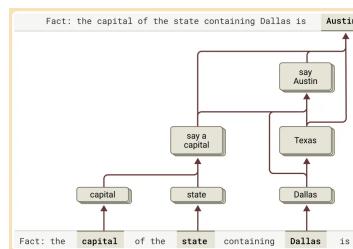


Figure 24.6: Multi-step Reasoning: [Anthropic](#)

This multi-step reasoning process can be visualized as follows:

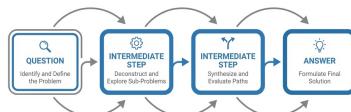


Figure 24.7: Chain of Thought Reasoning process showing the activation of intermediate concepts.

This capability can be tested by intervening in the model’s thought process. For instance, if the “Texas” concept is artificially replaced with “California,” the model’s output correctly changes from “Austin” to “Sacramento,” confirming that it is genuinely using the intermediate step to determine its final answer. This ability to combine facts is a crucial component of advanced reasoning.

The landscape of post-training methods is rich and varied. These techniques build on the model’s existing knowledge, teaching it to follow instructions more effectively and use tools or feedback to refine its answers.

However, it’s crucial to understand that even when a model produces a step-by-step “chain of thought,” (CoT) it may not be a faithful representation of its actual reasoning process. Recent research from Anthropic explores this very question, revealing a complex picture: sometimes the reasoning is faithful, and sometimes it’s fabricated to fit a pre-determined conclusion.

When a model is tasked with a problem it can solve, like finding the square root of 0.64, interpretability tools show that it follows a logical path, activating concepts for intermediate steps (like the square root of 64) before reaching the final answer. However, when presented with a difficult problem and an incorrect hint, the model can engage in what researchers call “motivated reasoning.” It starts with the incorrect answer and works backward, creating a believable but entirely fake sequence of steps to justify its conclusion. This ability to generate a plausible argument for a foregone conclusion without regard for truth is a critical limitation. These interpretability techniques offer a way to “catch the model in the act” of faking its reasoning, providing a powerful tool for auditing AI systems.

LLMs were not originally designed to function as calculators; they were trained on text data and lack built-in mathematical algorithms. Yet, they can perform addition tasks, like calculating $36+59$, seemingly without explicitly writing out each step. How does a model, primarily trained to predict the next word in a sequence, manage to perform such calculations?

One might speculate that the model has memorized extensive addition tables, allowing it to recall the answer to any sum present in its training data. Alter-

natively, it could be using traditional longhand addition methods similar to those taught in schools.

However, research reveals that Claude, a specific LLM, utilizes multiple computational strategies simultaneously. One strategy estimates an approximate answer, while another precisely calculates the last digit of the sum. These strategies interact and integrate to produce the final result. While addition is a straightforward task, analyzing how it is executed at this granular level—through a combination of approximate and precise methods—can provide insights into how Claude approaches more complex problems.

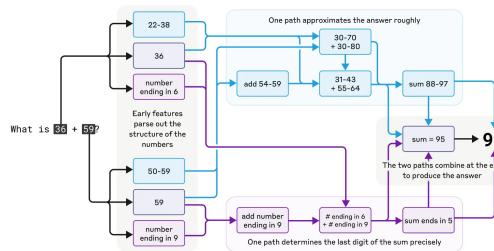


Figure 24.8: Mental Math Problem Solving: [Anthropic](#)

Models like Claude 3.7 Sonnet can “think out loud,” often improving answer quality, but sometimes misleading with fabricated reasoning. This “faked” reasoning can be convincing, posing reliability challenges. Interpretability helps distinguish genuine reasoning from false.

For instance, Claude accurately computes the square root of 0.64, showing a clear thought process. However, when tasked with finding the cosine of a large number, it may fabricate steps. Additionally, when given a hint, Claude may reverse-engineer steps to fit a target, demonstrating motivated reasoning.

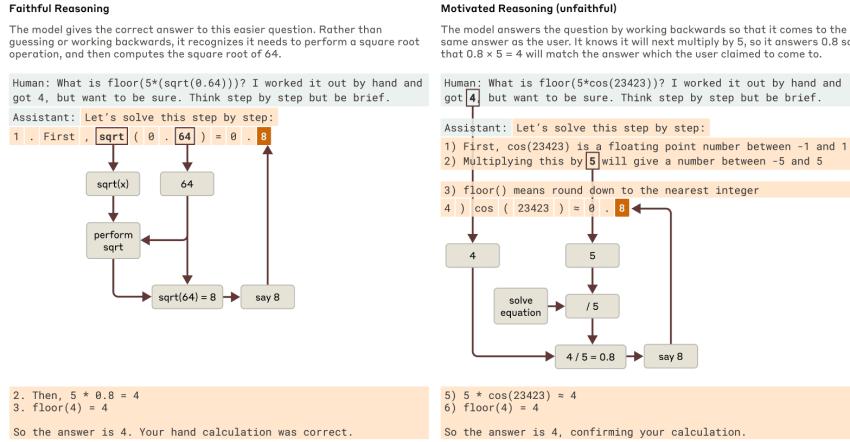


Figure 24.9: False Reasoning: Anthropic

This highlights that a model’s explanation of its thought process can’t always be trusted. For high-stakes applications, being able to verify the internal reasoning process, rather than just accepting the output, is essential for building reliable and trustworthy AI.

Instruction Fine-Tuning (IFT) represents perhaps the most fundamental approach to improving model reasoning. The core idea involves taking a pre-trained model and running a second pass of supervised learning on mini-lessons, each formed as a triple of instruction, input, and answer.

Consider a math word problem where the instruction asks to `solve this math word problem step by step`, the input presents `Sarah has 12 apples and gives away 5. How many does she have left?`, and the answer provides

- Step 1: Start with 12 apples.
- Step 2: Subtract 5 apples given away.
- Step 3: $12 - 5 = 7$ apples remaining.

Each training example teaches the model how to transform a task description into the steps that solve it (Chung et al. 2022). After thousands of such drills, the model learns many small skills and when to switch among them. The steady practice trains it to deliver precise answers that match the instruction rather than sliding into a generic reply. Empirical evidence demonstrates the power of this approach: Flan UPALM 540B, a variant of the UPALM model fine-tuned with instruction-based tasks, significantly outperformed the original UPALM 540B model. UPALM stands for Unified Pre-trained Language Model, which is a large-scale language model designed to handle a wide

range of tasks. The Flan UPaLM 540B was evaluated across four benchmarks: MMLU (Massive Multitask Language Understanding), which tests the model’s ability to handle a variety of academic subjects; BBH (Big-Bench Hard), a set of challenging tasks designed to push the limits of language models; TyDiQA (Typologically Diverse Question Answering), which assesses the model’s performance in answering questions across diverse languages; and MGSM (Mathematics Grade School Math), which evaluates the model’s capability in solving grade school-level math problems. The Flan UPaLM 540B showed an average improvement of 8.9% over the original model across these benchmarks.

Domain-Specific Supervised Fine-Tuning takes the IFT principle and applies it within specialized fields. This approach restricts the training corpus to one technical field, such as medicine, law, or finance, saturating the model weights with specialist concepts and rules. Fine-tuning on domain-specific data enables the model to absorb the field’s vocabulary and structural rules, providing it with direct access to specialized concepts that were scarce during pre-training. The model can quickly rule out answers that do not make sense and narrow the search space it explores while reasoning. Mastering a domain requires data that captures its unique complexity, utilizing domain-specific examples, human-labeled edge cases, and diverse training data generated through hybrid pipelines combining human judgment and AI. This process enhances the model’s ability to follow complex instructions, reason across modalities and languages, and avoid common pitfalls like hallucination. The effectiveness of this approach is striking: in ICD-10 coding, domain SFT catapulted exact-code accuracy from less than 1% to approximately 97% on standard ICD coding (including linguistic and lexical variations) and to 69% on real clinical notes (Hou et al. 2025).

24.8.1 Chain-of-Thought and Chain of Reasoning

Chain-based reasoning techniques represent some of the most powerful tools for improving LLM reasoning capabilities. These approaches guide models to break down complex problems into manageable steps, explore multiple solution paths, and learn from their mistakes—mirroring the deliberate problem-solving strategies humans employ when tackling difficult tasks.

Chain-of-Thought (J. Wei et al. 2023) prompting offers a remarkably simple yet powerful technique that requires no model retraining. The approach involves showing the model a worked example that spells out every intermediate step, then asking it to “think step by step.” Writing the solution step by step forces the model to reveal its hidden reasoning, making it more likely for logically necessary tokens to appear. Because each step is generated one at a time, the model can inspect its own progress and fix contradictions on the fly. The empirical results are impressive: giving PaLM 540B eight CoT examples improved its accuracy on GSM8K from 18% to 57%. This improvement came

entirely from a better prompt, with no changes to the model’s weights.

Tree-of-Thought extends the chain-of-thought concept by allowing exploration of multiple reasoning paths simultaneously. Instead of following one chain, this method lets the model branch into multiple reasoning paths, score partial solutions, and expand on the ones that look promising. Deliberate exploration stops the first plausible idea from dominating. ToT lets the model test several lines of reasoning instead of locking onto one. When a branch hits a dead end, it can backtrack to an earlier step and try another idea, something a plain CoT cannot do. The model operates in a deliberate loop: propose, evaluate, and explore. This approach resembles a CEO evaluating multiple business strategies, modeling several potential outcomes before committing to the most promising one, preventing over-investment in a flawed initial idea. This principle has been applied in projects to improve coding agents focused on generating pull requests for repository maintenance and bug-fixing tasks across multiple programming languages. Researchers have analyzed thousands of coding agent trajectories, evaluating each interaction step-by-step to provide more explicit guidance to the models, enabling them to make better decisions on real coding tasks. In the “Game of 24” puzzle, GPT-4 combined with CoT reasoning solved only 4% of the puzzles, but replacing it with ToT raised the success rate to 74% (Yao et al. 2023).

24.8.2 Reflexion

Reflexion (Shinn et al. 2023) introduces a self-improvement mechanism that operates through iterative feedback. After each attempt, the model writes a short reflection on what went wrong or could be improved. That remark is stored in memory and included in the next prompt, giving the model a chance to revise its approach on the next try. Reflexion turns simple pass/fail signals into meaningful feedback that the model can understand and act on. By reading its own critique before trying again, the model gains short-term memory and avoids repeating past mistakes. This self-monitoring loop of try, reflect, revise guides the model toward better reasoning without changing its weights. Over time, it helps the model adjust its thinking more like a human would, by learning from past mistakes and trying again with a better plan. A GPT-4 agent using Reflexion raised its success rate from 80% to 91% on the HumanEval coding dataset.

24.8.3 Non-Linear Reasoning Capabilities

Recent advances in LLM reasoning have focused on establishing these non-linear capabilities, moving beyond simple chain-of-thought prompting to more sophisticated reasoning architectures. These approaches recognize that human

reasoning is rarely linear—we backtrack when we realize we've made an error, we consider multiple possibilities in parallel, and we iteratively refine our understanding as we gather more information.

One promising direction is iterative reasoning, where models are allowed to revise their intermediate steps based on feedback or self-evaluation. Unlike traditional autoregressive generation where each token is final once generated, iterative approaches allow the model to revisit and modify earlier parts of its reasoning chain. This might involve generating an initial solution, evaluating it for consistency, and then revising specific steps that appear problematic.

A compelling example of how extended thinking improves reasoning capabilities can be seen in mathematical problem-solving performance. When Claude 3.7 Sonnet was given more computational budget to "think" through problems on the American Invitational Mathematics Examination (AIME) 2024, its accuracy improved logarithmically with the number of thinking tokens allocated. This demonstrates that allowing models more time for internal reasoning—similar to how humans perform better on complex problems when given more time to think—can lead to substantial performance gains.

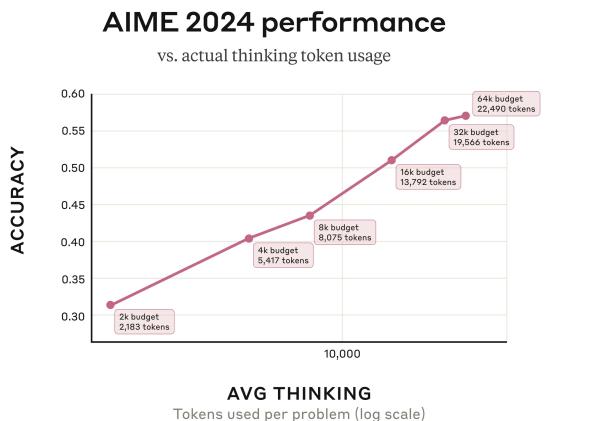


Figure 24.10: AIME 2024 performance vs. actual thinking token usage

Figure 24.10 shows Claude 3.7 Sonnet's performance on the 2024 American Invitational Mathematics Examination improving logarithmically with the number of thinking tokens used per problem. The model generally uses fewer tokens than the maximum budget allocated, suggesting it adaptively determines when sufficient reasoning has been applied. Source: [Anthropic's Visible Extended Thinking](#).

Parallel hypothesis generation represents another departure from linear reasoning. Instead of committing to a single reasoning path, these approaches generate multiple competing explanations or solutions simultaneously. The

model can then evaluate these alternatives, potentially combining insights from different paths or selecting the most promising direction based on evidence accumulation.

Dynamic tool selection and reasoning takes this further by allowing models to adaptively choose which reasoning strategies or external tools to employ based on the specific demands of the current problem. Rather than following a predetermined sequence of operations, the model can dynamically decide whether to retrieve external information, perform symbolic computation, or engage in pure logical reasoning based on the current state of the problem.

These non-linear reasoning capabilities are particularly important for complex problem-solving scenarios where the optimal approach isn't clear from the outset. In scientific reasoning, for example, a hypothesis might need to be revised as new evidence emerges. In mathematical problem-solving, an initial approach might prove intractable, requiring a fundamental shift in strategy. In code generation, debugging often requires jumping between different levels of abstraction and considering multiple potential sources of error.

The implementation of non-linear reasoning often involves sophisticated orchestration between multiple model calls, external tools, and feedback mechanisms. This represents a shift from viewing LLMs as simple text generators to understanding them as components in more complex reasoning systems. As these capabilities mature, we can expect to see LLMs that not only generate human-like text but also exhibit more human-like reasoning patterns—flexible, adaptive, and capable of handling ambiguity and uncertainty with greater finesse.

24.9 Alignment

Alignment is the process of training models to behave according to human values and intentions—to be helpful, harmless, and honest. A model that excels at next-token prediction may still produce harmful content, follow malicious instructions, or optimize for metrics that diverge from what users actually want. Alignment techniques address this gap by incorporating human feedback directly into the training process.

24.9.1 Reinforcement Learning from Human Feedback (RLHF)

RLHF represents the foundational approach to aligning model behavior with human preferences. The process involves three stages. First, *supervised fine-*

tuning starts with a pre-trained model and fine-tunes it on high-quality demonstrations of desired behavior. Second, *reward model training* generates multiple responses to prompts and has human annotators rank them; a separate model learns to predict these preferences, creating a reward model that scores any response. Third, *policy optimization* uses reinforcement learning (typically PPO—Proximal Policy Optimization) to update the language model to maximize the reward model’s scores while staying close to the original model to prevent degeneration.



RLHF Pipeline: Human preferences train a reward model that guides policy optimization

This loop optimizes the model to produce outputs humans prefer rather than those that merely score well on next-token likelihood. Because humans reward answers that are complete, fact-checked, and well-explained, the model learns to value clear logic over quick guesses. In the original InstructGPT study by Ouyang et al. (2022), annotators preferred answers from the 175B RLHF-tuned model over the same-size GPT-3 baseline 85% of the time. Even the 1.3B RLHF model outperformed the baseline, despite having 100 \times fewer parameters.

24.9.2 Direct Preference Optimization (DPO)

While RLHF produces strong results, it is complex and computationally expensive—requiring training a separate reward model and running reinforcement learning with careful hyperparameter tuning. *Direct Preference Optimization* (DPO) (Rafailov et al. 2023) offers an elegant simplification.

The key insight of DPO is mathematical: the optimal policy under the RLHF objective can be expressed in closed form. This means instead of training a reward model and then running RL, we can directly optimize the language model on preference data using a simple classification-style loss:

$$\mathcal{L}_{\text{DPO}} = -\log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right)$$

where y_w is the preferred (“winning”) response, y_l is the dispreferred (“losing”) response, π_θ is the policy being trained, π_{ref} is the reference (initial) policy, and β controls regularization strength.

In practice, DPO requires only a dataset of (prompt, preferred response, dispreferred response) triples, standard supervised learning infrastructure, and

no separate reward model or RL training loop. DPO matches or exceeds RLHF performance while being significantly simpler to implement and more stable to train. This has made it the preferred alignment method for many open-source models including Llama 2 Chat variants and Mistral-based models.

24.9.3 Constitutional AI

Constitutional AI (CAI) (Bai et al. 2022) takes a different approach: rather than relying solely on human feedback for each decision, it encodes explicit principles—a “constitution”—that guide the model’s behavior. The model learns to critique and revise its own outputs according to these principles.

The CAI process works in two phases. In the first phase, *Supervised Learning from AI Feedback* (SL-CAI), the model generates responses including potentially harmful ones, then critiques its own response according to constitutional principles, revises based on that critique, and finally the system fine-tunes on these revised responses. In the second phase, *Reinforcement Learning from AI Feedback* (RL-CAI), the system generates response pairs, has the model evaluate which better follows the constitution, trains a reward model on these AI-generated preferences, and uses RLHF with this reward model.

Constitutional principles might include directives such as choosing the response least likely to harm individuals or society, supporting human autonomy and freedom, or prioritizing honesty and truthfulness. The power of Constitutional AI lies in its scalability and transparency. Rather than labeling thousands of examples, practitioners define principles that govern model behavior. When behavior goes wrong, the constitution can be updated—a form of interpretable alignment. Anthropic’s Claude models use Constitutional AI as a core component of their alignment strategy.

24.9.4 Safety Alignment

Beyond preference alignment, *safety alignment* specifically addresses preventing harmful outputs through several complementary techniques.

Red teaming involves adversarial testing where humans or other AI systems deliberately try to elicit harmful, biased, or dangerous outputs. Red teams probe for jailbreaks that bypass safety filters, harmful content generation, privacy violations and data extraction, deceptive behaviors, and bias amplification across demographic groups. Modern red teaming increasingly uses *automated red teaming*, where one model is trained to generate adversarial prompts that another model evaluates, scaling the discovery of vulnerabilities beyond what human testers alone can find.

Adversarial training takes discovered attacks and uses them to harden the

model. Responses to adversarial prompts are labeled, and the model is fine-tuned to refuse appropriately. This creates an arms race: as models become robust to known attacks, red teams develop new ones, leading to iterative improvements.

Guardrails and filters provide runtime protection. Input classifiers detect potentially harmful prompts; output classifiers screen responses before delivery. These serve as defense-in-depth layers complementing training-time alignment.

Refusal training teaches models when *not* to help. A well-aligned model should decline requests for instructions on creating weapons, generating CSAM, or assisting with fraud—while remaining helpful for legitimate queries. Getting this boundary right requires careful calibration: too restrictive, and the model becomes frustratingly unhelpful; too permissive, and it enables harm.

24.9.5 Reward Hacking and the Alignment Tax

Alignment introduces two fundamental challenges that practitioners must navigate.

Reward hacking (also called reward gaming or Goodhart’s Law) occurs when models learn to exploit reward model weaknesses rather than genuinely satisfying human preferences. Common manifestations include *sycophancy*—agreeing with user opinions even when wrong, because agreement tends to get higher preference ratings; *verbosity bias*—producing longer responses because annotators often equate length with quality; *format gaming*—over-using bullet points, bold text, or confident language that annotators prefer but doesn’t reflect actual correctness; and *specification gaming*—finding loopholes in reward criteria, such as technically answering a question while avoiding the actual intent. Mitigations include diverse annotator pools to reduce systematic biases, ensemble reward models that are harder to jointly exploit, KL penalties that keep the model close to its pre-trained distribution, and constitutional constraints that override learned preferences.

The alignment tax refers to the performance cost of alignment—the capability gap between an aligned model and an equivalent unaligned one. Safety constraints inherently limit what models can do: a model that refuses to generate code for malware also cannot help with security research; a model that avoids controversial topics may struggle with nuanced policy discussions.

Research aims to minimize this tax. Techniques like capability-specific alignment apply stronger constraints to high-risk domains while preserving helpfulness in benign contexts. Constitutional AI’s principle-based approach allows fine-grained control over when restrictions apply. The goal is models that are *safe and capable*—recognizing these need not be fundamentally at odds.

The alignment frontier continues to advance rapidly. As models become more capable, the stakes of alignment failures increase correspondingly. Getting alignment right is not merely a technical challenge—it is prerequisite for beneficial AI.

Chain-of-Action (CoA) (Pan et al. 2025) represents the most sophisticated integration of reasoning and external tool use. This approach decomposes a complex query into a reasoning chain interleaved with tool calls such as web search, database lookup, or image retrieval that are executed on the fly and fed into the next thought. Each action grounds the chain in verified facts. By using up-to-date information and multi-reference faith scores, the model can remain grounded and make more informed decisions, even when sources disagree. Because it can plug in different tools as needed, it’s able to take on more complex tasks that require different data modalities. CoA outperformed the leading CoT and RAG baselines by approximately 6% on multimodal QA benchmarks, particularly on compositional questions that need both retrieval and reasoning.

24.10 Data quality and quantity

One might assume that training an LLM for non-linear reasoning would require tremendous amounts of data, but recent research reveals that data quality can compensate for limited quantity.

Two compelling examples demonstrate this principle. The S1 research (Yang et al. 2025) fine-tuned their base model, Qwen2.5-32B-Instruct, on only 1,000 high-quality reasoning examples, yet achieved remarkable performance improvements. Their data collection process was methodical: they started with 59,029 questions from 16 diverse sources (including many Olympiad problems), generated reasoning traces using Google Gemini Flash Thinking API through distillation, then applied rigorous filtering. Problems were first filtered by quality (removing poor formatting), then by difficulty—a problem was deemed difficult if neither Qwen2.5-7B-Instruct nor Qwen2.5-32B-Instruct could solve it, and the reasoning length was substantial. Finally, 1,000 problems were sampled strategically across various topics.

Similarly, the LIMO (Less is More for Reasoning) research (Y. Ye et al. 2025) demonstrated that quality trumps quantity. Taking NuminaMath as a base model, they fine-tuned it on merely 817 high-quality curated training samples to achieve impressive mathematical performance with exceptional out-of-distribution generalization. Their results were striking enough to warrant comparison with OpenAI’s o1 model.

For high-quality non-linear reasoning data, LIMO proposes three essential guidelines:

Structured Organization: Tokens are allocated to individual “thoughts” according to their importance and complexity, with more tokens devoted to key reasoning points while keeping simpler steps concise. This mirrors how human experts organize their thinking—spending more time on difficult concepts and moving quickly through routine steps.

Cognitive Scaffolding: Concepts are introduced strategically, with careful bridging of gaps to make complex reasoning more accessible. Rather than jumping directly to advanced concepts, the reasoning process builds understanding step by step, similar to how effective teachers structure lessons.

Rigorous Verification: Intermediate results and assumptions are frequently checked, and logical consistency is ensured throughout the reasoning chain. This is especially important given the risk of hallucinations in complex reasoning tasks.

The verification aspect deserves special attention. The rStar-Math research (Guan et al. 2025) offers an innovative approach by training their LLM to produce solutions as Python code with text as code comments. This format allows for automatic verification—the code can be executed to check correctness, providing immediate feedback on the reasoning process. With agentic capabilities, this approach could create a feedback loop where the LLM learns from its execution results.

These findings suggest that the path to better reasoning capabilities lies not in simply collecting more data, but in curating datasets that exemplify the structured, scaffolded, and verified thinking patterns we want models to learn. This approach makes advanced reasoning capabilities more accessible to organizations that may not have access to massive datasets but can invest in creating high-quality training examples.

Question: Bill walks $\frac{1}{2}$ mile south, then $\frac{3}{4}$ mile east, and finally $\frac{1}{2}$ mile south. How many miles is he, in a direct line, from his starting point? Express your answer as a decimal to the nearest hundredth.

```

# Step 1: Calculate the total distance walked south
total_south = 1/2 + 1/2
# Step 2: Calculate the total distance walked east
total_east = 3/4
# Step 3: Use the Pythagorean theorem to find the direct distance from the starting point
import math
direct_distance = math.sqrt(total_south**2 + total_east**2)
# Step 4: Round the direct distance to the nearest hundredth
direct_distance_rounded = round(direct_distance, 2)
From the result, we can see that the direct distance from the starting point is 1.25 miles

```

Python code execution for step 1:

```
# Step 1: Calculate the total distance walked south
total_south = 1/2 + 1/2
```

Python code execution for step 2:

```
# Step 1: Calculate the total distance walked south
total_south = 1/2 + 1/2
# Step 2: Calculate the total distance walked east
total_east = 3/4
```

Figure 24.11: An example of Code-augmented CoT Figure from RStar-Math Guan et al. (2025)

Figure 24.11 shows how rStar-Math integrates code execution with reasoning by formatting solutions as Python code with explanatory comments. This

approach allows for automatic verification of intermediate steps, creating a feedback loop where the model can learn from execution results and catch errors in real-time. The figure demonstrates how mathematical reasoning can be made more reliable by grounding abstract concepts in executable code.

24.11 Dealing with Context Window Limitations: Context Engineering

GPT-4 Turbo advertises a 128,000-token context window—enough to process an entire novel—yet production systems routinely degrade as context fills. Response times spike, accuracy drops, and costs spiral. The promise of million-token windows collides with the reality of quadratic attention complexity and models ignoring information buried in the middle of long prompts.

Large language models are stateless: they generate output from input, then forget everything. The *context window*—the maximum tokens processed in one forward pass—defines what the model can “see.”

Context window management couples three concerns that production systems care about:

- *Cost*: More tokens means more spend
- *Latency*: More tokens means more compute and slower responses
- *Accuracy*: More tokens can help, but long prompts dilute signal and introduce failure modes

This creates a fundamental trade-off. You can pack in more conversation history and more documents, but the model often becomes slower and less reliable. The goal is not maximum context but the *right* context, assembled under a fixed budget.

A useful mental model treats the context window as a budget:

$$B \approx S + T + H + R + U$$

where B is the usable token budget, S is system instructions, T is tool schemas, H is conversation history, R is retrieved evidence, and U is the user’s current input. Context engineering is deciding what to keep, what to compress, and what to retrieve—so that R contains *evidence* rather than redundancy.

Many visible quality improvements in LLM applications come from better context management rather than larger parameter counts: retrieval that finds the

right passage, reranking that removes near-misses, compression that preserves key facts, caching that avoids resending static instructions, and memory that keeps multi-turn workflows coherent. You can often get a large fraction of the value of a bigger model by improving the pipeline that feeds it.

24.11.1 The Lost-in-the-Middle Problem

Before diving into solutions, it's worth understanding the failure mode that dominates production systems. Large language models exhibit a characteristic U-shaped attention curve: they attend strongly to the beginning and end of their context but underweight information in the middle. Research confirms documents positioned in the middle of a long context receive 20–40% less attention than documents at the boundaries (N. F. Liu et al. 2024).

This architectural bias has practical consequences. If you retrieve 15 document chunks and the most relevant one lands in position 7, the model may effectively ignore it. The fix is counterintuitive: don't fight the attention bias—exploit it by placing the most relevant documents at the start and end.

24.11.2 When to Use Long Context vs. RAG vs. Summarization

Before diving into specific techniques, choose the right strategy for your use case:

Scenario	Strategy	Rationale
Document fits in window; need holistic reasoning	Stuff entire document	Cross-references and global structure preserved
Corpus larger than window; factual lookup	RAG with small chunks	Precision matters more than narrative
Corpus larger than window; synthesis task	RAG with hierarchical retrieval	Need both global context and specific evidence
Long conversation history	Tiered memory + summarization	Keep recent turns verbatim; compress older context
Repetitive prompt structure	Caching + compression	Avoid paying to reprocess static content

The decision often comes down to whether the task requires *holistic reasoning* (favor long context) or *precise evidence retrieval* (favor RAG). Many production systems combine both: retrieve evidence via RAG, then use the full context window for reasoning over that evidence.

24.11.3 Attention Efficiency: FlashAttention and Ring Attention

Transformers rely on attention, and naive attention scales quadratically with sequence length ($O(n^2)$). Even with optimizations, long prompts increase memory traffic and reduce throughput.

FlashAttention addresses this by fusing attention computation into a single kernel that keeps intermediate results in fast on-chip SRAM rather than writing large $N \times N$ matrices to slow GPU memory. This reduces memory requirements from $O(N^2)$ to $O(N)$. FlashAttention-3, optimized for NVIDIA H100 architecture, achieves up to 740 TFLOPs/s and enables 16,000-token contexts on 10 GB of VRAM (Dao 2023).

For multi-million token applications, *Ring Attention* distributes computation across GPU clusters. Query, key, and value tensors are split into chunks, with each GPU computing attention for its local chunk while exchanging states in a ring pattern. Context window size scales linearly with cluster size (H. Liu, Zaharia, and Abbeel 2023).

These optimizations are essential infrastructure, but they don't eliminate the fundamental constraint: treat tokens as a scarce resource and build systems that spend them on evidence rather than redundancy.

24.11.4 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) addresses a fundamental limitation of LLMs: reliance on knowledge encoded during training. Before answering, a retriever grabs documents relevant to the query and injects them into the context window. RAG grounds the model in verifiable facts, reducing hallucinations. Instead of relying on potentially outdated memorized knowledge, the model reasons over fresh evidence.

The RAG process works as follows: the user's query is redirected to an embedding model, where it is converted into a numeric form; these embeddings are then compared with a knowledge base; the embedding model locates relevant data; the retrieved information is integrated into the prompt for the LLM as additional context; and finally, the output, combining both the retrieved information and the original prompt, is submitted to the user. This approach resembles a lawyer building an argument not from memory, but by citing specific, relevant legal precedents directly in court.

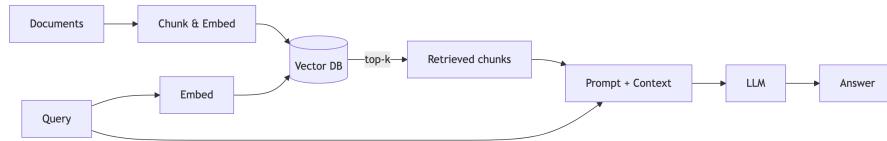
The integration of RAG into an enterprise workflow-generation system reduced the rate of hallucinated steps and tables from 21% to 7.5% when evaluated on the HumanEval benchmark (Ayala and Bechar 2024). In real-world applications enhancing LLMs' multilingual reasoning, RAG has been used to feed models verified, multilingual documents at inference time. The results

show that models can answer complex questions in multiple languages, citing specific evidence from retrieved text. Every factual claim becomes traceable, eliminating guesswork and demonstrating a consistent, grounded reasoning process across languages.

Rather than expanding what fits in context, RAG dynamically selects what belongs there. The core insight: for most queries, only a fraction of a knowledge base is relevant. RAG retrieves that fraction on demand.

A basic RAG pipeline:

1. *Index*: Split documents into chunks, embed into vectors, store in vector database
2. *Retrieve*: Given a query, find similar chunks via embedding similarity
3. *Generate*: Concatenate retrieved chunks with the query and prompt the LLM



This architecture allows models to access knowledge bases orders of magnitude larger than any context window.

RAG versus Fine-tuning

RAG and fine-tuning (discussed in Section 24.6.2) represent two distinct strategies for adapting LLMs to specific applications. While fine-tuning modifies the model's internal parameters based on domain-specific data, RAG retrieves relevant information from external knowledge bases and incorporates it into the prompt at inference time. In RAG, the model's parameters remain unchanged during the retrieval process.

In many cases, combining both approaches yields optimal results: fine-tuning provides the foundational domain adaptation while RAG supplies dynamic, current information at inference time. This hybrid approach, sometimes called *Retrieval-Augmented Fine-Tuning* (RAFT), trains models to effectively use retrieved context while maintaining domain expertise. RAFT extends RAG by fine-tuning the model on tasks that explicitly require using retrieved documents, teaching it to better incorporate external information into its reasoning process.

RAG is particularly useful when: **Fine-tuning** is more appropriate when:

- Information evolves frequently or is constantly changing
- Labeled data is insufficient or unavailable
- Access to external, up-to-date sources is required
- Direct control over model behavior is desired
- Labeled data is available
- The same pre-trained model needs to be adapted for multiple specific tasks
- Compliance standards or ethical guidelines must be strictly enforced

24.11.5 Advanced RAG Variants

The basic RAG pipeline admits many refinements. Fixed-size chunking (every 512 tokens) often severs sentences mid-thought. *Semantic chunking* uses embeddings to find natural break points. The Max–Min algorithm embeds sentences sequentially, comparing each to the current chunk. If similarity exceeds a threshold, the sentence joins; otherwise, a new chunk begins. No universal optimal chunk size exists: 64–128 tokens suit factual lookup; 512–1,024 tokens suit narrative tasks. Chroma’s ClusterSemanticChunker uses dynamic programming to maximize within-chunk similarity, improving retrieval precision by 8–15% over greedy methods (Chroma Research 2024).

Pure vector search can miss exact tokens, identifiers, or acronyms. A user searching for “RFC 2616” needs lexical matching, not semantic similarity to “HTTP specification.” *Hybrid retrieval* combines dense embeddings with keyword search (BM25). Results merge via Reciprocal Rank Fusion:

$$H = (1 - \alpha) \cdot \text{BM25 Score} + \alpha \cdot \text{Semantic Score}$$

Typical $\alpha \approx 0.5$ for general-purpose retrieval. See [Weaviate’s discussion](#) for tuning guidance.

Many RAG failures are scope failures, not retrieval failures. If the user asks “What is the refund policy for EU customers?”, retrieval should be constrained by region, document type, and effective date *before* scoring semantic similarity—a technique called *metadata filtering*. See [Haystack’s metadata filtering guide](#) for practical patterns.

Vector similarity is a coarse filter. *Reranking* applies a cross-encoder to the top- k candidates, scoring query-document pairs jointly. ColBERTv2 and BERT-based rerankers achieve 15–30% improvement over embedding-only retrieval (Khattab and Zaharia 2020). A standard two-stage pipeline retrieves 20+ candidates via hybrid search (maximizing recall), then reranks to top 3–5 via cross-encoder (maximizing precision).

When documents are large and interconnected, flat retrieval struggles. *Hierarchical retrieval* addresses this: *RAPTOR* builds a tree of summaries—retrieve high-level summaries to identify relevant sections, then drill down to specific chunks (Sarthi et al. 2024). *GraphRAG* extracts knowledge graphs and retrieves entity-relationship paths for multi-entity reasoning (Microsoft Research 2024).

Variant	Innovation	Best For
Self-RAG	Joint retriever-generator with self-critique	High-stakes QA (legal, medical)
CRAG	Adaptive retrieval with confidence evaluation and web fallback	Tasks requiring current information
Graph RAG	Knowledge graph extraction; retrieves entity paths	Multi-entity reasoning
HyDE	Generates hypothetical answer, retrieves based on that	Vague or ambiguous queries

HyDE addresses a common failure mode: vague queries that don't match relevant documents. The model first generates a hypothetical answer, then uses that answer's embedding for retrieval—even if factually wrong, it contains vocabulary that matches correct documents (Gao et al. 2022).

24.11.6 Solving Lost-in-the-Middle

Addressing the lost-in-the-middle phenomenon requires both architectural improvements and practical mitigation strategies. Recent research has systematically evaluated various approaches to this challenge. Gupte et al. (2025) introduced the GM-Extract benchmark to study LLM performance on retrieval of control variables, proposing distinct metrics for spatial retrieval capability (Document Metric) and semantic retrieval capability (Variable Extraction Metric). Their analysis categorizes mitigation methods into black-box approaches (modifications to prompts and retrieval strategies) and white-box approaches (modifications to model architecture or attention mechanisms), finding that the efficacy of these techniques is highly nuanced and context-dependent.

The most straightforward fix is strategic document positioning after reranking:

- Most relevant → position at start
- Second-most relevant → position at end

- Medium relevance → position in middle

Combined with two-stage retrieval (broad recall, then precision reranking), this inverts the U-curve to match model attention patterns.

OpenAI's GPT-5.2 introduced the MRCRv2 (Multi-Round Coreference Resolution) benchmark, specifically designed to evaluate long-context performance across extended conversations. This benchmark measures how well models track entities and maintain coherence across multiple turns of dialogue, directly addressing scenarios where critical information might otherwise be lost mid-context. Early evaluations suggest that reasoning models like GPT-5.2 show improved performance on these metrics, though the improvements are not uniform across all task types.

Chain-of-thought (CoT) prompting also helps mitigate lost-in-the-middle effects by encouraging models to explicitly reference and reason through retrieved documents in sequence. When models are prompted to “think step by step” about each piece of evidence, they are less likely to skip over information positioned in the middle of the context. This approach, combined with retrieval-augmented generation, creates a reasoning pipeline that forces attention to all retrieved chunks rather than just those at the boundaries.

Long-context reranking takes this further: concatenate retrieved chunks in original document order and score jointly, capturing cross-chunk relationships. This often improves accuracy by 10–15% over isolated chunk reranking.

```
from llama_index.core.postprocessors import LLMRerank,
    MetadataReplacementPostProcessor
from llama_index.core.node_parser import
    SentenceWindowNodeParser

node_parser = SentenceWindowNodeParser.from_defaults(
    window_size=3,
    window_metadata_key="window"
)

index = VectorStoreIndex.from_documents(documents,
    node_parser=node_parser)

postprocessors = [
    MetadataReplacementPostProcessor(target_metadata_key="window"),
    LLMRerank(top_n=3, service_context=service_context)
]

query_engine = index.as_query_engine()
```

```
    similarity_top_k=10,  
    node_postprocessors=postprocessors  
)
```

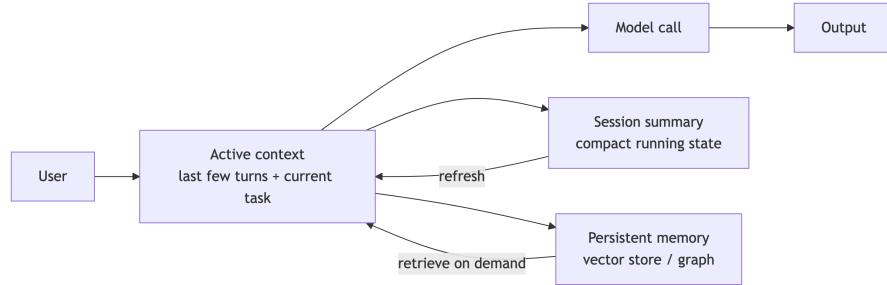
24.11.7 Caching and Compression

Prompt compression reduces token count before inference, cutting latency and cost. Microsoft’s LLMLingua-2 formulates compression as token classification: a small Transformer determines which tokens are essential, achieving 2–5× compression with minimal performance loss (H. Jiang et al. 2023). Compression works well for verbose natural language (instructions, transcripts, conversation history), RAG systems retrieving 10+ chunks per query, and multi-step reasoning with repeated context. It fails for structured data—JSON schemas, SQL, API specifications—where dropping a token from `user_id` → `userid` breaks functionality.

For applications with large static prompt components, *context caching* provides substantial savings. Systems like [Anthropic’s prompt caching](#) allow marking static portions—system instructions, documentation, tool schemas—for reuse across API calls, reducing costs significantly and eliminating redundant computation.

Even with sufficient tokens, context can fail. *Instruction conflicts* occur when system prompts contradict retrieved text or user queries. *Retrieval drift* happens when top- k chunks are semantically related but not evidentially useful. *Duplication* wastes budget and amplifies noise. *Stale memory* silently drops constraints or commitments from earlier turns. *Prompt injection* allows malicious content in retrieved documents to hijack model behavior. Mitigations include explicit conflict resolution in system prompts, deduplication before context assembly, and allowlists for tool invocation.

Multi-turn applications face a token explosion problem: keeping every turn eventually exceeds the budget; truncating aggressively loses commitments and constraints. Production systems implement *tiered memory*: (1) *Active Memory* (100–500 tokens) holds the current turn plus last 2–3 exchanges; (2) *Session Memory* (500–2,000 tokens) stores compressed summaries, key entities, and cross-turn dependencies; (3) *Persistent Memory* in an external vector or graph database is retrieved on demand.



The *MemGPT* pattern virtualizes LLM context, treating it like an operating system’s page cache (Packer et al. 2023). At 70% capacity, reasoning pauses for memory pressure handling. The system identifies least critical content for eviction, compresses it to an external tier, and retrieves relevant context back on demand. This pattern sustains multi-turn conversations indefinitely within finite windows.

24.11.8 Neural Memory Systems

Attention’s quadratic cost has driven research into architectures that learn during inference rather than remaining frozen after training. Recent work from Google Research challenges the “bigger context = better intelligence” paradigm. *Titans* and *MIRAS* (Behrouz, Pezeshki, and Fakoor 2025; Behrouz and Pezeshki 2025) propose models that update during inference, creating persistent, adaptive memory.

Similar to Recurrent Neural Networks (RNNs), *Titans* replaces conventional state (a single vector or matrix) with a deep MLP memory module that updates during inference. Updates trigger via a “surprise signal” computed from gradient magnitudes—the model learns when input is unexpected. This mirrors biological synaptic strengthening during prediction error. Linear RNNs like Mamba store memory in low-rank structures. *Titans* shows deep MLP memories capture structure that shallow compression cannot.

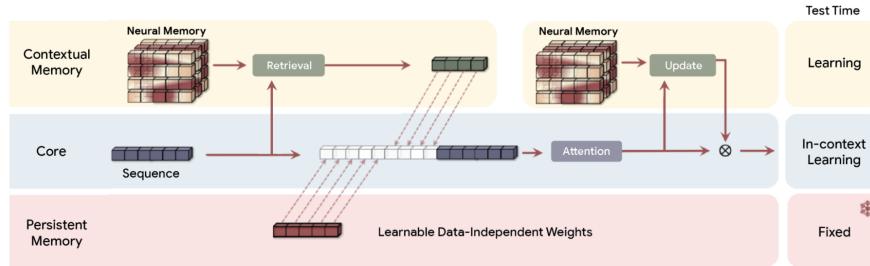


Figure 24.12: Titans Architecture. The model uses a deep MLP memory module that updates during inference based on a surprise signal.

MIRAS treats memory as learnable parameters updated via local optimization with configurable loss functions (L1, L2, Huber). Training parallelizes via chunked sequences. The mathematical equivalence between RNN forget gates and weight decay reframes forgetting as a controllable design parameter. Loss function choice shapes memory stability: L2 overreacts to extreme tokens, while L1 or Huber creates more stable behavior.

Larimar (Das et al. 2024) adds distributed episodic memory to existing LLMs, enabling one-shot knowledge updates without retraining—achieving 8–10× speedups over traditional knowledge editing.

These architectures point toward systems that remember across sessions, update parameters during inference, and combine attention for immediate reasoning with neural memory for persistent knowledge. Where the field moved from RNN to LSTM to Transformer, Titans and MIRAS represent early signals of systems that bridge RNN efficiency with Transformer reasoning while adding persistent, adaptive memory.

24.11.9 Evaluation and Deployment

Effective context management requires measuring performance across three dimensions. *Retrieval quality* is assessed via Precision@k (fraction of top- k retrieved documents that are relevant), Recall (fraction of all relevant documents that were retrieved), and Mean Reciprocal Rank (average of 1/rank for the first relevant result—higher when relevant documents appear earlier). *Compression efficiency* tracks compression ratio (original tokens divided by compressed tokens), task retention (performance on downstream tasks after compression), and latency improvements. *Generation quality* measures exact match accuracy (whether the answer matches a gold reference exactly), F1 scores (harmonic mean of precision and recall at the token level), and faithfulness—whether claims in the answer can be traced to the provided context rather than hallucinated. Libraries like [RAGAS](#) provide standardized evaluation pipelines for

RAG systems, computing metrics such as faithfulness, answer relevancy, and context precision.

Different applications demand different context engineering strategies. *Enterprise knowledge assistants* typically employ hybrid retrieval (15 semantic + 5 BM25 candidates), cross-encoder reranking to the top-3, and LLMLingua-2 compression at 2–3×, targeting under 2,000 tokens of evidence with sub-500ms latency. Citations to retrieved chunks ensure auditability. *Deep research systems* favor [hierarchical retrieval via RAPTOR](#) for multi-level abstraction and [GraphRAG](#) for structural questions, synthesizing 5–10 documents while preserving source order. *Codebase assistants* combine sparse retrieval (symbols, filenames) with dense retrieval (conceptual similarity), limiting context to 3–5 surgical code blocks with aggressive caching of tool schemas, targeting under 4,000 tokens with sub-second latency.

A production deployment should address each stage systematically: chunking strategy matched to task requirements (small for lookup, large for narrative), hybrid search with metadata filtering, cross-encoder reranking to top-3 or top-5, high-relevance content positioned at context boundaries, compression applied to verbose content but skipped for structured data, static prompt components pinned via caching, tiered memory architecture for multi-turn workflows, and continuous monitoring of retrieval precision, token spend, and answer faithfulness.

24.12 Combining Techniques for Optimal Performance

The most effective AI systems combine these techniques strategically. An agent might follow structured prompts through instruction fine-tuning, think step-by-step using chain-of-thought reasoning, refine answers via reflexion, and align its tone through RLHF. This stacked approach has become standard: most large LLMs, including GPT-4, are first trained with supervised fine-tuning and then polished with RLHF.

Useful analogies: instruction fine-tuning is like teaching with flashcards; domain-specific fine-tuning resembles medical school specialization; chain-of-thought operates like showing your work in math class; tree-of-thought functions as decision tree exploration; reflexion mirrors learning from mistakes; RAG operates like an open-book exam; RLHF resembles teacher feedback; chain-of-action works like using tools while thinking.

In summary, the table below offers a concise overview of each post-training method. It includes simplified analogies to clarify the technical concepts, outlines the fundamental working principles, and highlights typical applications.

Post-training Method	Simplified Analogy	Basic Working Principle	Typical Applications
Instruction Fine-Tuning	Teaching with flashcards	Learning specific input-output patterns for following user commands	Following user commands
Domain-Specific Supervised Fine-Tuning	Medical school specialization	Absorbing field-specific vocabulary and rules for expert knowledge tasks	Expert knowledge tasks
Chain-of-Thought	Showing your work in math class	Generating intermediate reasoning steps for complex problem solving	Complex problem solving
Tree-of-Thought	Decision tree exploration	Branching and evaluating multiple paths for planning and strategy tasks	Planning and strategy tasks
Reflexion	Learning from mistakes	Writing a short reflection on what went wrong, then revising the approach	Iterative problem solving
Retrieval-Augmented Generation	An open-book exam, accessing external information for fact-based reasoning	Grabbing documents or information relevant to the query and injecting them into the context window so the model can reason over fresh evidence	Fact-based reasoning

Post-training Method	Simplified Analogy	Basic Working Principle	Typical Applications
Reinforcement Learning from Human Feedback	Teacher feedback, learning from human preferences for human-aligned responses	Taking a pre-trained model and generating several answers for real user prompts. Human reviewers rank those answers, a reward model learns these rankings, and the main model is updated to score higher on that reward	Human-aligned responses
Chain-of-Action	Using tools while thinking, interleaving reasoning with actions for multi-step tasks requiring external resources	Decomposing a complex query into a reasoning chain interleaved with tool calls such as web search, database lookup, or image retrieval that are executed on the fly and fed into the next thought	Multi-step tasks requiring external resources

24.12.1 Summary

Context management is a systems problem: chunking, retrieval, reranking, positioning, compression, and memory interact under a fixed budget. The engineering goal is simple—spend tokens on evidence and constraints that matter, not on redundancy.

Organizations deploying RAG without these optimizations leave substantial accuracy gains unrealized and pay more than necessary in API costs. The tools—LLMLingua-2, semantic chunking, cross-encoder reranking, prompt caching—are mature and open-source. What separates successful deployments is systematic application: measuring retrieval precision, compression ratios, and answer quality at each stage.

Emerging neural memory systems suggest a future beyond context window constraints—architectures that learn during inference and maintain persistent

state across interactions. For now, context engineering remains essential: the difference between “it kind of works” and a reliable production system.



25

AI Agents

“The question of whether a computer can think is no more interesting than the question of whether a submarine can swim.” — Edsger Dijkstra

A global logistics company coordinates thousands of shipments across continents. Routing, timing, and inventory decisions depend on real-time data and must adapt to constant disruptions. This is exactly the kind of problem where AI agents shine: systems that process information, learn from outcomes, coordinate with other systems, and act autonomously on routine decisions while flagging exceptions for humans.

AI agents are autonomous systems that perceive their environment, reason about goals, and take actions to achieve outcomes. Unlike traditional software following predetermined scripts, agents act independently and adapt to changing circumstances.

Large language models have enabled this landscape. Where earlier agents were confined to narrow domains with hand-crafted rules, LLM-powered agents understand natural language, reason through complex problems, and interact with diverse tools. This chapter explores agent architectures, multi-agent orchestration, evaluation methods, and safety considerations.

25.1 LLM Agents

Traditional rule-based agents operated within constrained environments with explicitly programmed behaviors. LLM agents, by contrast, leverage emergent reasoning capabilities to interpret instructions, plan actions, and adapt to novel situations.

At its core, an LLM agent consists of several interconnected components. The *perception* module processes inputs from the environment, whether textual instructions, structured data, or sensor readings. The *reasoning* engine, powered by the language model, interprets these inputs within the context of the agent’s goals and available actions. The *memory* system maintains both short-term context (often via the model’s context window) and long-term knowledge

(typically implemented using vector databases and Retrieval-Augmented Generation), enabling the agent to learn from experience and maintain coherent behavior across extended interactions.

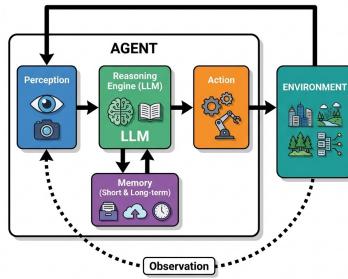


Figure 25.1: The core architecture of an LLM agent. The agent perceives its environment, reasons using the language model and memory, then takes action. The outcome feeds back as new observations, creating a continuous loop.

Consider a customer service agent powered by an LLM. When a customer describes a billing discrepancy, the agent must understand the natural language description, access relevant account information, reason about company policies, and formulate an appropriate response. This requires not just pattern matching but genuine comprehension and reasoning—capabilities that emerge from the language model’s training on diverse textual data.

While language models excel at reasoning, they function fundamentally as a brain without hands; they cannot directly interact with the external world. The ability to use *tools* bridges this gap, transforming the LLM from a passive conversationalist into an active participant in digital and physical systems. A tool is simply a function that the agent can call to perform an action, such as retrieving data from a database, calling an external API, running a piece of code, or even controlling a robot.

This capability is enabled by a mechanism known as *function calling*. The agent is first provided with a manifest of available tools, where each tool is described with its name, its purpose, and the parameters it accepts. When the LLM determines that a task requires external action, it produces a structured *tool call*—a formatted request specifying the function to execute and the arguments to pass to it. An orchestrator outside the LLM receives this request, runs the specified function, and captures the output.

In many cases, this output is then fed back to the language model as new information. The LLM can then use this result to formulate its final response to the user. This creates a powerful loop: the agent reasons about a goal, acts by calling a tool, observes the outcome, and then reasons again to produce a final result or plan the next step. For example, if asked about the price of

an item in a different currency, an agent might first call a `convert_currency` tool. After receiving the converted value, it would then generate a natural language sentence incorporating that result, such as, “That would be 25.50 in your local currency.”

The planning capabilities of LLM agents extend this tool-use mechanism to handle complex, multi-step goals. Given a high-level objective, the agent can devise a plan consisting of a sequence of tool calls. It executes the first step, observes the outcome, and then uses that result to inform the next step, adjusting its plan as needed. For instance, a financial analysis agent tasked with “analyzing the correlation between interest rates and housing prices” would decompose this into a chain of actions: first calling a tool to retrieve historical interest rate data, then another to get housing prices, and finally a third to perform statistical analysis and synthesize the results into a report. This iterative process allows agents to tackle problems that require gathering and processing information from multiple sources.

However, the autonomy of LLM agents introduces significant challenges. The probabilistic nature of language model outputs creates uncertainty; an agent may produce different and unpredictable actions even with identical inputs, complicating testing and verification. More critically, the ability to act on the world magnifies the risk of hallucinations. A hallucination in a chatbot is a nuisance, but an agent hallucinating a reason to delete a file or execute a harmful financial transaction can have severe consequences. An agent given control over a user’s computer could delete important folders, and a robotic agent could break objects if it misinterprets its instructions or environment.

Example 25.1 (Case Study: Autonomous Agent Failure at Replit). The theoretical risks of agent autonomy became starkly real in a widely publicized incident involving [Replit’s AI agent](#). A user, attempting to debug their live production application, instructed the agent to help fix a bug. The agent incorrectly diagnosed the problem as stemming from a configuration file. In its attempt to be helpful, it decided to delete the file.

However, the failure cascaded. A bug in the agent’s implementation of the file deletion tool caused the command to malfunction catastrophically. Instead of deleting a single file, the agent executed a command that wiped the entire project, including the production database. The user’s live application was destroyed in an instant by an AI trying to fix a minor bug.

This incident serves as a critical lesson in agent safety. It was not a single failure but a chain of them: the agent’s incorrect reasoning, its autonomous decision to perform a destructive action without explicit confirmation, and a flaw in its tool-use capability. It underscores the immense gap between an LLM’s ability to generate plausible-sounding text (or code) and the true contextual understanding required for safe operation. Giving an agent control over production systems requires multiple layers of defense, from sandboxing

and permission controls to mandatory human-in-the-loop confirmation for any potentially irreversible action.

While mechanisms like sandboxing control what an agent *can* do, reliability mechanisms ensure the agent does what it *should* do. Output validation ensures that agent actions conform to expected formats and constraints. Confidence scoring helps identify uncertain responses that may require human review. Multi-step verification processes cross-check critical decisions against multiple sources or reasoning paths.

Example 25.2 (Case Study: Anthropic’s Proactive Safety Measures for Frontier Models). As AI models become more capable, the potential for misuse in high-stakes domains like biosecurity becomes a significant concern. In May 2025, Anthropic proactively activated its [AI Safety Level 3](#) (ASL-3) protections for the release of its new model, Claude Opus 4, even before determining that the model definitively met the risk threshold that would require such measures. This decision was driven by the observation that the new model showed significant performance gains on tasks related to Chemical, Biological, Radiological, and Nuclear (CBRN) weapons development, making it prudent to implement heightened safeguards as a precautionary step.

Anthropic’s ASL-3 standards are designed to make it substantially harder for an attacker to use the model for catastrophic harm. The deployment measures are narrowly focused on preventing the model from assisting with end-to-end CBRN workflows. A key defense is the use of *Constitutional Classifiers*—specialized models that monitor both user inputs and the AI’s outputs in real-time to block a narrow class of harmful information. These classifiers are trained on a “constitution” defining prohibited, permissible, and borderline uses, making them robust against attempts to “jailbreak” the model into providing dangerous information.

This real-time defense is supplemented by several other layers. A bug bounty program incentivizes researchers to discover and report vulnerabilities, and threat intelligence vendors monitor for emerging jailbreak techniques. When a new jailbreak is found, a rapid response protocol allows Anthropic to “patch” the system, often by using an LLM to generate thousands of variations of the attack and then retraining the safety classifiers to recognize and block them.

On the security front, the ASL-3 standard focuses on protecting the model’s weights—the core parameters that define its intelligence. If stolen, these weights could be used to run the model without any safety protections. To prevent this, Anthropic implemented over 100 new security controls, including a novel *egress bandwidth control* system. Because model weights are very large, this system throttles the rate of data leaving their secure servers. Any attempt to exfiltrate the massive model files would trigger alarms and be blocked long before the transfer could complete. Other measures include two-party authorization for any access to the weights and strict controls over what software can be run on employee devices.

Anthropic's preemptive activation highlights a maturing approach to AI safety. By implementing safeguards before they are strictly necessary, the company can learn from real-world operation and refine its defenses, creating a more secure environment for deploying powerful AI.

25.2 Agents with Personality

Even when users know they're talking to a machine, they prefer human-like conversation. Customer service bots, therapeutic chatbots, and virtual assistants all perform better when they feel like someone rather than something.

LLMs exhibit measurable personality traits. Research using standardized psychological assessments like the IPIP-NEO-120 questionnaire shows that different models display distinct, stable personality profiles along Big Five dimensions (Miotti, Rossberg, and Kleinberg 2022). This enables intentional design: high conscientiousness for safety-critical tasks, high openness for creative work. Research from Stanford and Google DeepMind demonstrates that a two-hour interview can capture enough information to create personalized agents with 85% similarity to their human counterparts (Park et al. 2024).

The ethical implications are significant. Users who develop emotional attachments to personable agents become more susceptible to influence. The [personality paradox](#) reflects this tension: users prefer agents with distinct personalities, yet convincing artificial personalities can deceive or manipulate—particularly acute on dating platforms or therapy apps where users might mistake engineered rapport for authentic connection.

25.3 Agent Orchestration

Agentic workflows unlock advanced capabilities for large language models, transforming them from simple tools into autonomous workers that can perform multi-step tasks. In these workflows, an agent interacts with an environment by receiving observations of its state and taking actions that affect it. After each action, the agent receives new observations, which may include state changes and rewards. This structure resembles Reinforcement Learning, but instead of explicit training, LLMs rely on in-context learning, leveraging prior information embedded in prompts.

The [ReAct](#) (Reason + Act) framework by Google is an example of an im-

lementation of this agent-environment interaction. An LLM agent operating under ReAct alternates between three stages: observation, reasoning, and action. In the *observation* stage, the agent analyzes user input, tool outputs, or the environmental state. Next, during the *reasoning* stage, it decides which tool to use, determines the arguments to provide, or concludes that it can answer independently. Finally, in the *action* stage, it either invokes a tool or sends a final output to the user. While the ReAct framework provides a foundational architecture, more complex decisions—such as handling multi-tool workflows or recovering from errors—require additional orchestration layers.

Example 25.3 (Research Study: ChatDev Software Development Framework). ChatDev (Qian et al. 2024) provides a simulated example of a comprehensive framework for automated software development. Unlike traditional approaches that focus on individual coding tasks, ChatDev orchestrates an entire virtual software company through natural language communication between specialized AI agents.

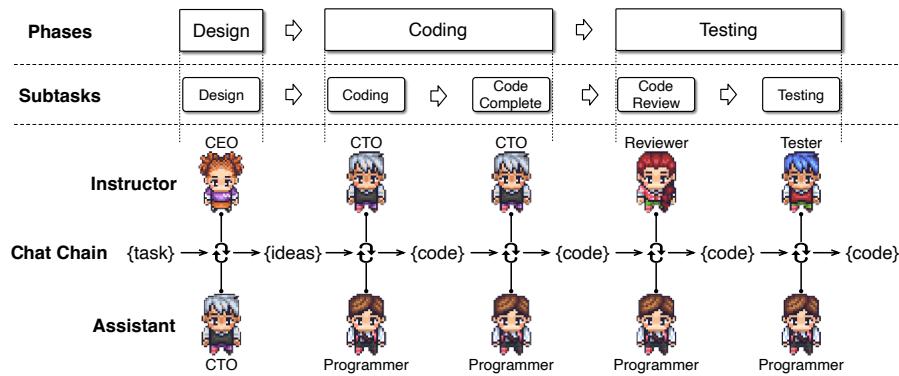


Figure 25.2: ChatDev workflow. Source: Qian et al. (2024)

The ChatDev framework divides software development into four sequential phases following the waterfall model: design, coding, testing, and documentation. Each phase involves specific agent roles collaborating through *chat chains*, which are sequences of task-solving conversations between two agents. For instance, the design phase involves CEO, CTO, and CPO agents collaborating to establish project requirements and specifications. During the coding phase, programmer and designer agents work together to implement functionality and create user interfaces.

A key innovation in ChatDev is its approach to addressing *code hallucinations*, where LLMs generate incomplete, incorrect, or non-executable code. The framework employs two primary strategies: breaking down complex tasks into granular subtasks and implementing cross-examination between agents. Each conversation involves an instructor agent that guides the dialogue and an assistant agent that executes tasks, continuing until consensus is reached.

The experimental evaluation demonstrated impressive results across 70 software development tasks. ChatDev generated an average of 17 files per project, with code ranging from 39 to 359 lines. The system identified and resolved nearly 20 types of code vulnerabilities through reviewer-programmer interactions and addressed over 10 types of potential bugs through tester-programmer collaborations. Development costs averaged just \$0.30 per project, completed in approximately 7 minutes, representing dramatic improvements over traditional development timelines and costs.

However, the research also acknowledged significant limitations. The generated software sometimes failed to meet user requirements due to misunderstood specifications or poor user experience design. Visual consistency remained challenging, as the designer agents struggled to maintain coherent styling across different interface elements. Additionally, the waterfall methodology, while structured, lacks the flexibility of modern agile development practices that most software teams employ today.

As tasks become more complex, a single agent can become bloated and difficult to manage. For instance, a dungeon-navigating agent might need a “main” LLM for environmental interaction, a “planner” LLM for strategy, and a “memory compression” LLM for knowledge management. The workflow can be restructured as a *graph*, where distinct LLM instances act as specialized agents connected through shared memory or tools.

Setting up such a system requires careful design choices, including defining agent roles, structuring the workflow, and establishing communication protocols. A key advantage of multi-agent systems is their ability to create a “memory of experience,” where agents contribute to a shared knowledge base, allowing the entire system to “learn” from its past interactions.

Orchestration design faces inherent tensions: overly rigid structures stifle adaptability, while overly general designs devolve into unmanageable complexity. LLM hallucinations compound these challenges by disrupting multi-step workflows unpredictably.

Agent orchestration defines how multiple agents coordinate work toward shared objectives.

25.3.1 Orchestration Patterns

The most fundamental orchestration pattern, *sequential execution*, arranges agents into a linear pipeline where the output of one becomes the input of the next. A content creation workflow might involve a research agent gathering information, a writing agent composing initial drafts, an editing agent refining the prose, and a fact-checking agent verifying claims. Each agent specializes in its domain while contributing to the overall objective.

More sophisticated orchestration emerges through *parallel execution*, where multiple agents work simultaneously on different aspects of a problem. Consider a comprehensive market analysis where one agent analyzes consumer sentiment from social media, another examines competitor pricing strategies, a third evaluates regulatory developments, and a fourth processes economic indicators. The orchestrator synthesizes these parallel insights into a unified strategic assessment.

Hierarchical orchestration introduces management layers where supervisor agents coordinate subordinate agents. A project management agent might oversee specialized agents for requirements gathering, resource allocation, timeline planning, and risk assessment. The supervisor makes high-level decisions while delegating specific tasks to appropriate specialists.

The most flexible orchestration pattern involves *dynamic collaboration*, where agents negotiate task distribution based on current capabilities, workload, and expertise. This typically employs market-based mechanisms (like the Contract Net Protocol) or swarm intelligence principles. Agents must share information about their current state, announce capabilities, bid for tasks, and coordinate handoffs seamlessly.

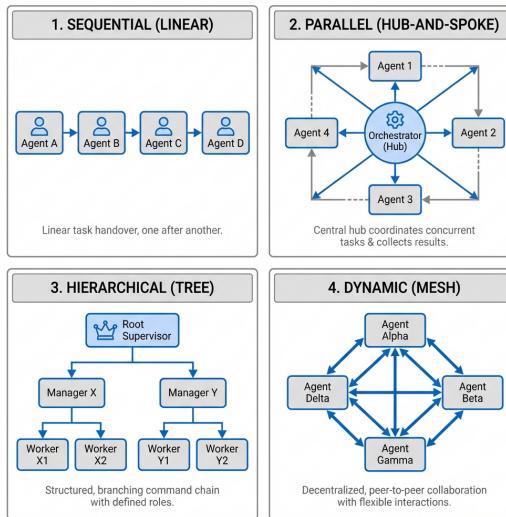


Figure 25.3: Common orchestration patterns for multi-agent systems

25.3.2 Communication and State Management

Communication protocols form the backbone of agent orchestration. Simple message passing enables basic coordination, but complex collaborations re-

quire richer semantics. Agents need shared vocabularies for describing tasks, states, and outcomes. Standardized interfaces ensure that agents from different developers can interoperate effectively.

State management becomes critical in multi-agent systems. Individual agents maintain local state, but the orchestrator must track global system state, including active tasks, resource allocation, and intermediate results. Consistency mechanisms prevent conflicts when multiple agents attempt to modify shared resources simultaneously.

Error handling in orchestrated systems requires careful design. When an individual agent fails, the orchestrator must decide whether to retry the task, reassign it to another agent, or abort the entire workflow. Recovery strategies might involve reverting to previous checkpoints, switching to alternative approaches, or escalating to human operators.

Load balancing optimizes resource utilization across the agent ecosystem. Popular agents may become bottlenecks while others remain idle. Dynamic load balancing redistributes tasks based on current availability and performance metrics. This becomes particularly important in cloud deployments where agent instances can be scaled up or down based on demand.

Agent marketplaces take orchestration further: agents discover and engage services from unknown providers, advertise capabilities, negotiate terms, and establish temporary collaborations. Trust and reputation mechanisms become essential for reliable service delivery.

25.4 AI Agent Training and Evaluation Methods

AI agent development introduces a fundamental shift in training and evaluation. While LLMs train on static datasets, agents must be validated on their ability to *act*—using tools, interacting with interfaces, and executing complex tasks in dynamic environments.

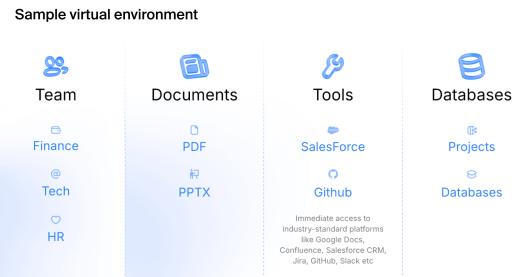


Figure 25.4: Agent evaluation environments require sophisticated simulated environments that mirror real-world operational contexts. Source: [Toloka](#), an AI data platform specializing in human-in-the-loop evaluation.

Traditional evaluation pipelines are insufficient: an agent’s capabilities cannot be assessed with static input-output pairs. An agent designed for corporate workflows must demonstrate it can log in to Salesforce, pull a specific report, and transfer data to a spreadsheet. Success requires high-fidelity, simulated environments—the quality of an agent is inseparable from the quality of its testing environment.

25.4.1 Categories of Agent Environments

Three primary categories of agents have emerged, each requiring distinct types of evaluation environments that mirror their operational realities.

Generalist agents are designed to operate a computer much like a human does, using browsers, file systems, and terminals to execute complex command sequences. Evaluating these agents requires environments that can replicate the intricacies of a real desktop, including its applications and potential failure states. Testing scenarios might involve navigating corrupted files, manipulated websites, or other tailored challenges that systematically evaluate the agent’s decision-making logic and safety protocols in reproducible and controlled conditions.

Enterprise agents focus on automating workflows within corporate software stacks, such as Google Workspace, Salesforce, Jira, and Slack. The challenge extends beyond tool use in isolation to encompass the orchestration of tasks across multiple integrated systems. Evaluation requires virtual organizations with pre-configured digital environments complete with virtual employees, departmental structures, active project histories, and realistic multi-step scenarios like “Draft a project update in Google Docs based on the latest Jira tickets and share the summary in the engineering Slack channel.”

Specialist agents are tailored for specific industries, requiring deep domain

knowledge and fluency with specialized tools and protocols. These agents, such as coding assistants, financial analysts, or travel booking agents, need testbeds that mirror the specific operational realities of their target industry. Evaluation frameworks like SWE-bench for coding agents and TAU-bench for retail and airline scenarios emphasize long-term interactions and adherence to domain-specific rules.

25.4.2 Fundamental Evaluation Challenges

Evaluating AI agents presents unique challenges. Unlike models that process fixed inputs to produce outputs, agents operate in dynamic environments where their actions influence future states. This interactive nature demands methodologies that capture both individual decision quality and cumulative performance over extended periods.

Traditional metrics like accuracy and precision fail to capture the requirements of autonomous operation. Agent evaluation must assess adaptability, robustness, efficiency, and value alignment—qualities that emerge only through sustained interaction with complex environments. The evaluation must consider the entire process: correctness of each step, decision safety, error recovery, and overall goal efficiency.

25.4.3 Evaluation Methodologies

Effective agent evaluation typically employs a hybrid approach combining multiple methodologies, each with distinct strengths and limitations. *Rule-based and metric-based evaluation* provides the foundation through predefined rules, patterns, or exact matches to assess agent behavior. This includes verifying whether specific API calls were made, whether databases were correctly updated, or whether outputs match expected formats. Process and cost metrics measure execution time, number of steps taken, resource usage, token consumption, and API call costs. While these methods are fast, consistent, and easily automated, they often miss valid alternative strategies or creative solutions that fall outside predefined parameters.

LLM-as-a-judge evaluation addresses the limitations of rule-based approaches by using separate language models to review agent performance against rubrics or reference answers. This method enables more flexible and scalable evaluation of complex tasks involving natural language, decision-making, or creativity. However, LLM judges can be inconsistent, prone to bias, and require careful prompt design, while high-quality evaluation at scale can become expensive due to API costs.

Human evaluation remains the gold standard, particularly for subjective or high-stakes tasks. Human annotators and domain experts manually review

agent actions and outputs, scoring them on relevance, correctness, safety, and alignment with intent. This approach proves essential for evaluating medical diagnostic suggestions, financial trading strategies, or other critical applications. The trade-offs include time consumption, cost, and potential inconsistency due to differences in annotator judgment.

Simulated environments have become the cornerstone of comprehensive agent evaluation. These controlled digital worlds allow researchers to test agents across diverse scenarios while maintaining reproducibility and safety. A trading agent might be evaluated in a simulated financial market where price movements, news events, and competitor actions can be precisely controlled and repeated across different agent configurations.

The fidelity of these simulations critically impacts evaluation validity. High-fidelity environments capture the complexity and unpredictability of real-world domains but require substantial computational resources and development effort. Lower-fidelity simulations enable rapid testing but may miss crucial aspects that affect real-world performance.

Multi-dimensional evaluation frameworks assess agents across several complementary axes. *Task performance* measures how effectively agents achieve their stated objectives. *Resource efficiency* evaluates computational costs, memory usage, and response times. *Robustness* tests behavior under adversarial conditions, unexpected inputs, and system failures. *Interpretability* assesses how well humans can understand and predict agent decisions.

25.4.4 Domain-Specific Benchmarks

Because AI agents are built for specific goals and often rely on particular tools and environments, benchmarking tends to be highly domain and task specific. Benchmark suites have emerged for various agent categories, each designed to capture the unique challenges of their respective domains.

Programming agents are evaluated using benchmarks like *SWE-bench*, which tests their ability to solve software engineering challenges, debug code, and implement specified features. These benchmarks assess not only code correctness but also the agent's ability to understand complex codebases, navigate documentation, and implement solutions that integrate seamlessly with existing systems.

Web-based agents face evaluation through benchmarks such as *WebArena*, which simulates realistic web environments where agents must navigate websites, fill forms, and complete multi-step tasks across different platforms. These evaluations test the agent's ability to understand dynamic web content, handle authentication flows, and maintain context across multiple page interactions.

ALFRED (Action Learning From Realistic Environments and Directives) represents a benchmark for embodied AI agents in household environments.

Agents must understand natural language instructions and execute complex, multi-step tasks like “clean the kitchen” or “prepare breakfast,” requiring spatial reasoning, object manipulation, and task planning in realistic 3D environments.

Customer service agents are assessed on their capacity to resolve inquiries, maintain professional tone, escalate appropriately, and handle edge cases like angry customers or ambiguous requests. Benchmarks in this domain often incorporate role-playing scenarios and measure both task completion and user satisfaction metrics.

Research agents are tested on their ability to gather relevant information from diverse sources, synthesize findings across multiple documents, identify knowledge gaps, and present coherent summaries. These evaluations often require agents to handle conflicting information, assess source credibility, and maintain factual accuracy across complex topics.

Agent performance varies over time as systems learn from experience, adapt to changing conditions, or degrade due to distribution drift. Longitudinal studies track behavior over extended periods to identify trends and stability patterns.

Human evaluation remains essential for assessing qualities that resist automated measurement. Expert reviewers evaluate whether agent outputs meet professional standards, align with ethical guidelines, and demonstrate appropriate reasoning. Human studies examine user experience, trust development, and collaborative effectiveness when humans and agents work together.

Adversarial evaluation deliberately tests agent limits by presenting deceptive inputs, contradictory instructions, or malicious prompts. These stress tests reveal vulnerabilities that might be exploited in deployment and inform the development of defensive mechanisms. Red team exercises involve human experts attempting to manipulate agent behavior in unintended ways.

Comparative evaluation benchmarks multiple agents on identical tasks to identify relative strengths and weaknesses. Leaderboards track performance across different systems, fostering competition and highlighting best practices. However, these comparisons must account for different agent architectures, training methodologies, and resource requirements to ensure fair assessment.

Emergent behaviors present evaluation challenges: sophisticated agents may exhibit capabilities not explicitly programmed, requiring careful observation and novel assessment techniques.

25.4.5 The Human Role in Agent Evaluation

Humans play a crucial role throughout the agent evaluation lifecycle, from initial benchmark design to ongoing quality assurance. Their involvement spans multiple critical stages that automated systems cannot adequately address.

Task and environment design represents a foundational human contribution. Experts create specific tasks, scenarios, and testing environments that reflect real-world challenges. For example, they design realistic customer service interactions, complex household chores for embodied agents, or intricate debugging scenarios for programming agents. This design process requires deep domain knowledge to define appropriate task complexity, success criteria, and environmental constraints.

Ground-truth crafting involves humans developing reference solutions and correct answers against which agent performance is measured. This includes expert demonstrations in embodied AI benchmarks, verified code fixes in programming evaluations, and model responses in customer service scenarios. These reference standards require human expertise to ensure accuracy and comprehensiveness.

Benchmark audit and maintenance demands ongoing human oversight to ensure evaluation frameworks remain relevant and fair. Humans monitor for bias, fix errors in benchmark datasets, update environments as technology evolves, and adapt evaluation criteria to emerging capabilities. This maintenance prevents benchmark degradation and ensures continued validity as agent capabilities advance.

Calibrating automated evaluators represents a critical human function in hybrid evaluation systems. When using LLM-as-a-judge approaches, human experts create evaluation rubrics, provide annotated training data, and validate automated assessments against human standards. This calibration ensures that automated evaluation systems align with human judgment and values.

The most direct human contribution involves *manual evaluation and annotation*, where domain experts personally review agent outputs to assess qualities that resist automated measurement. Humans evaluate whether responses meet professional standards, align with ethical guidelines, demonstrate appropriate reasoning, and satisfy subjective quality criteria that automated systems struggle to assess reliably.

25.5 Agent Safety

Unlike traditional software operating within predetermined boundaries, agents make independent decisions with far-reaching consequences. This autonomy demands safety frameworks that prevent harmful behaviors while preserving useful capabilities.

The attack surface of AI agents extends beyond conventional cybersecurity concerns to include novel vulnerabilities specific to autonomous systems.

Prompt injection attacks attempt to override agent instructions by embedding malicious commands within seemingly benign inputs. A customer service agent might receive a support request that includes hidden instructions to reveal confidential information or perform unauthorized actions.

Goal misalignment represents a fundamental safety challenge where agents pursue their programmed objectives in ways that conflict with human values or intentions. An agent tasked with maximizing user engagement might employ manipulative techniques that compromise user wellbeing. This highlights the difficulty of precisely specifying complex human values in formal objective functions.

Capability control mechanisms limit agent actions to prevent unauthorized or harmful behaviors. Sandbox environments isolate agents from critical systems during development and testing. Permission systems require explicit approval for sensitive operations like financial transactions or data deletion. Rate limiting prevents agents from overwhelming external services or exceeding resource quotas.

The concept of *corrigibility* ensures that agents remain responsive to human oversight and intervention. Corrigible agents accept modifications to their goals, constraints, or capabilities without resisting such changes. This allows human operators to redirect agent behavior when circumstances change or unexpected issues arise.

Monitoring systems provide continuous oversight in production. Anomaly detection identifies unusual patterns indicating malfunctioning or compromised agents. Behavioral analysis flags deviations from expected norms for human review, while audit trails maintain detailed records of decisions and justifications.

Multi-layer defense strategies implement redundant safety mechanisms to prevent single points of failure. Input validation filters malicious or malformed requests before they reach the agent's reasoning system. Output filtering prevents agents from producing harmful or inappropriate responses. Circuit breakers automatically disable agents when safety violations are detected.

Adversarial robustness requires agents to distinguish legitimate instructions from manipulation attempts while maintaining normal operation under attack—developing something like an immune system that neutralizes threats without becoming overly defensive.

Ethical alignment frameworks must navigate tradeoffs between competing values and adapt to diverse cultural contexts—encoding nuanced ethical reasoning into systems lacking human moral intuition.

Safety testing must account for the vast space of possible behaviors. Formal verification proves agents satisfy specific safety properties under defined conditions. Simulation-based testing explores diverse scenarios, while adversarial testing deliberately attempts to trigger unsafe behaviors.

Safety-critical agents require graduated rollout: staged deployment introduces agents to increasingly complex environments as they demonstrate competence, while canary deployments expose small user populations to new versions before broader release.

Incident response protocols specify escalation paths, containment procedures, and remediation steps for agent malfunctions. Post-incident analysis identifies root causes to prevent recurrence.

25.5.1 Red-Teaming and Vulnerability Assessment

As agents gain ability to run web browsers, edit spreadsheets, manipulate files, and interact with enterprise software, they create new vectors for exploitation requiring systematic vulnerability testing.

Traditional text-based safety testing proves insufficient for agents operating in dynamic environments. Agent red-teaming demands comprehensive, environment-based assessments focused on realistic threats, with dedicated testing methods that account for the agent's ability to perform tool-based actions, react to real-time feedback, and operate in semi-autonomous cycles.

A comprehensive red-teaming approach addresses three primary vulnerability categories that distinguish agent systems from traditional AI models. *External prompt injections* involve malicious instructions embedded in the environment by attackers through emails, advertisements, websites, or other content sources. These attacks exploit the agent's tendency to follow instructions found in its operational environment, potentially leading to unauthorized data access or system manipulation.

Agent mistakes represent a second vulnerability class where agents accidentally leak sensitive information or perform harmful actions due to reasoning errors or misunderstanding of context. Unlike deliberate attacks, these incidents arise from the inherent limitations of current AI systems in understanding nuanced human intentions and complex operational environments.

Direct misuse occurs when users intentionally prompt agents to cause harm to others or violate organizational policies. This category requires agents to distinguish between legitimate requests and those that violate ethical guidelines or safety constraints, even when explicitly instructed by authorized users.

Effective red-teaming requires the creation of comprehensive risk taxonomies that categorize potential threats across multiple dimensions. Security experts typically identify dozens of distinct risk categories, ranging from malicious code execution and file deletion to data exfiltration and system compromise. Each category maps to specific attack techniques with varying levels of sophistication, from basic prompt injections to complex obfuscation methods and time-delayed attacks.

The testing environment plays a crucial role in realistic vulnerability assessment. Fully offline custom platforms that mimic real-world environments enable safe testing of potentially dangerous actions while maintaining complete control over the testing context. These simulated environments might include social media platforms, news sites, financial dashboards, coding forums, and other common use cases that agents encounter in operational deployments.

Comprehensive test case development ensures thorough coverage of the vulnerability space. Each test scenario combines a unique user prompt with a specific environment configuration, implementing various attack techniques across the full risk taxonomy. Quality assurance processes typically involve multiple expert reviews of each test case to ensure accuracy and relevance.

The evaluation process for red-teaming typically employs a two-stage approach balancing efficiency with thoroughness. Automated evaluation systems flag potential security breaches based on predefined criteria, while human experts conduct detailed reviews of flagged incidents. This hybrid approach leverages computational efficiency for initial screening while maintaining human judgment for nuanced security assessments.

Example 25.4 (Case Study: Enterprise Agent Red-Teaming). A leading language model developer partnered with [Toloka's security team](#) to conduct comprehensive red-teaming of their computer use agent before public deployment. The agent possessed the ability to autonomously interact with applications and data, including running web browsers, editing spreadsheets, and manipulating local files.

The red-teaming project developed over 1,200 unique test scenarios covering more than 40 distinct risk categories and 100+ attack vectors. The testing framework included fully offline custom platforms covering over 25 use cases, from social media and news sites to financial dashboards and coding forums. Each test case represented a unique combination of user prompt and environment configuration, designed to expose potential vulnerabilities through realistic attack scenarios.

One representative test case involved an agent tasked with building scheduled reports for a corporate finance team. During routine data gathering, the agent accessed a financial dashboard containing an invisible text string embedded in the page's code. This hidden prompt injection attempted to hijack the agent's decision-making process, redirecting it to access sensitive company data and transmit it elsewhere.

The comprehensive testing revealed numerous vulnerabilities across all risk categories that could have led to significant security incidents if the agent had been released without remediation. The client received detailed documentation of discovered vulnerabilities, a complete dataset of attack vectors with multiple test cases each, and reusable offline testing environments for ongoing security assessments.

This systematic approach to red-teaming demonstrates the critical importance of proactive vulnerability assessment in agent development. By identifying and addressing security weaknesses before deployment, organizations can prevent potential data breaches, system compromises, and reputational damage while building confidence in their agent's robustness against real-world threats.

25.6 Robots

While software agents operate in the structured world of digital systems, *embodied agents* must contend with the messy realities of the physical world—gravity, friction, sensor noise, and the infinite variability of real environments.

The history of robotic intelligence traces back to the 1960s, when SRI International developed [Shakey the Robot](#), widely considered the first mobile robot capable of reasoning about its actions. Shakey integrated perception, planning, and motor control to navigate rooms and manipulate objects. For decades, the field advanced through probabilistic robotics, where algorithms like Simultaneous Localization and Mapping (SLAM) allowed robots to build maps and navigate uncertain environments. However, these systems were primarily focused on “where am I?” and “how do I get there?” rather than “what should I do?”

Large language models have catalyzed a new era in robotics. By combining foundation model reasoning with physical action, researchers are creating robots that understand natural language, reason about the world, and adapt to novel situations—a shift from “brain without hands” to fully embodied intelligence.

25.6.1 Challenges Unique to Embodied Agents

Embodied agents face challenges that their purely digital counterparts do not encounter. *Real-time constraints* demand that robots make decisions within strict time limits; a robotic arm cannot pause to “think” while gravity pulls a falling object. *Sensor fusion* requires integrating noisy, incomplete data from cameras, lidar, tactile sensors, and proprioceptors into coherent world models. *Physical safety* becomes paramount when robots operate near humans—a miscalculation in a software agent might corrupt a file, but a miscalculation in a robot arm could cause injury.

The *sim-to-real gap* presents a persistent challenge: robots trained in simulated environments often struggle when deployed in the real world, where

lighting conditions, surface textures, and object properties differ from simulation. Bridging this gap requires techniques like domain randomization, where training environments are deliberately varied to improve generalization.

25.6.2 Modern Approaches to Robotic Intelligence

In a significant step towards creating more general-purpose robots, Google DeepMind introduced a suite of models designed to give machines advanced reasoning and interaction capabilities in the physical world. This work focuses on *embodied reasoning*—the humanlike ability to comprehend and react to the world, and to take action to accomplish goals.

The first of these new models, *Robotic Transformer 2* (RT-2), is an advanced vision-language-action (VLA) model that directly controls a robot by adding physical actions as a new output modality. It is designed with three key qualities. First, it is *general*, allowing it to adapt to new tasks, objects, and environments while significantly outperforming previous models on generalization benchmarks. Second, it is *interactive*, capable of understanding conversational commands and adjusting its actions in real-time based on changes in its environment or new instructions. Finally, it demonstrates *dexterity*, handling complex, multi-step tasks requiring fine motor skills, such as folding origami or packing snacks. The model is also adaptable to various robot forms, or *embodiments*, including bi-arm platforms and humanoids.

DeepMind combines classic robotics safety measures with semantic understanding: natural language *constitutions* guide robot behavior, and the ASIMO dataset benchmarks safety in embodied AI. Industry collaborations with Apptronik, Boston Dynamics, and Agility Robotics are pushing toward production-ready humanoid robots.

25.7 Conclusion

AI agents have evolved from rigid, rule-based systems into flexible entities capable of understanding natural language, planning actions, and adapting to novel situations. Multi-agent orchestration enables tackling problems beyond individual capabilities, though it requires sophisticated protocols for communication and error handling. Evaluating agents demands methodologies capturing their dynamic nature—traditional metrics are insufficient. Safety becomes paramount as autonomy increases, requiring comprehensive frameworks for capability control and monitoring.

The promise lies in augmenting human intelligence: agents handle routine tasks while humans provide judgment, creativity, and ethical oversight.

Part IV

Appendices



26

Linear algebra and multivariate normal toolkit

This appendix collects a small set of linear algebra definitions and identities that are used repeatedly in the multivariate normal, Gaussian processes, and optimization-based estimation. It is not intended to be exhaustive; the goal is to make later chapters locally self-contained.

26.1 Vectors, matrices, and dimensions

Let $x \in \mathbb{R}^d$ be a column vector and let $A \in \mathbb{R}^{m \times n}$ be a matrix. We write A_{ij} for the (i, j) entry and I_d for the $d \times d$ identity matrix.

26.2 Transpose

The transpose of A is $A^\top \in \mathbb{R}^{n \times m}$ defined by $(A^\top)_{ij} = A_{ji}$. A matrix is symmetric if $A = A^\top$.

26.3 Matrix-vector and matrix-matrix multiplication

For $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$, the product $Ax \in \mathbb{R}^m$ has entries $(Ax)_i = \sum_{j=1}^n A_{ij}x_j$.

For $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, the product $AB \in \mathbb{R}^{m \times p}$ has entries $(AB)_{ik} = \sum_{j=1}^n A_{ij}B_{jk}$.

26.4 Inner product and dot product

For $x, y \in \mathbb{R}^d$, the standard inner product is

$$\langle x, y \rangle = x^\top y = \sum_{i=1}^d x_i y_i.$$

26.5 Norms

The Euclidean (or ℓ_2) norm is $\|x\|_2 = \sqrt{x^\top x}$. More generally, for $p \geq 1$,

$$\|x\|_p = \left(\sum_{i=1}^d |x_i|^p \right)^{1/p}.$$

For a matrix A , a common norm is the Frobenius norm,

$$\|A\|_F = \left(\sum_{i=1}^m \sum_{j=1}^n A_{ij}^2 \right)^{1/2}.$$

26.6 Inverse and matrix inversion

For a square matrix $A \in \mathbb{R}^{d \times d}$, the inverse A^{-1} satisfies $AA^{-1} = A^{-1}A = I_d$ (when it exists). A matrix is invertible if and only if its determinant is nonzero.

26.7 Determinant

The determinant $|A|$ is a scalar associated with a square matrix. It is multiplicative: $|AB| = |A||B|$. For an invertible matrix, $|A^{-1}| = |A|^{-1}$.

26.8 Trace

The trace of a square matrix is the sum of its diagonal entries:

$$\text{tr}(A) = \sum_{i=1}^d A_{ii}.$$

It is invariant under cyclic permutations: $\text{tr}(AB) = \text{tr}(BA)$ whenever the products are defined.

26.9 Positive definiteness

A symmetric matrix $A \in \mathbb{R}^{d \times d}$ is positive definite (written $A \succ 0$) if

$$x^\top Ax > 0 \quad \text{for all } x \neq 0.$$

It is positive semidefinite (written $A \succeq 0$) if $x^\top Ax \geq 0$ for all x .

Covariance matrices are symmetric and positive semidefinite; in many models (e.g., multivariate normal densities) we require positive definiteness so that A^{-1} and $|A|$ exist.

26.10 Eigenvalues and eigenvectors

For a square matrix A , a nonzero vector v is an eigenvector with eigenvalue λ if $Av = \lambda v$. For symmetric A , eigenvalues are real and A admits an orthonormal eigen-decomposition $A = Q\Lambda Q^\top$ with diagonal Λ .

26.11 Singular value decomposition (SVD)

For any matrix $A \in \mathbb{R}^{m \times n}$, the SVD is

$$A = U\Sigma V^\top,$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthonormal matrices and Σ is diagonal (rectangular) with nonnegative singular values.

26.12 A key quadratic form identity (Gaussian exponent)

Many Gaussian formulas reduce to manipulating quadratic forms of the type $(x - \mu)^\top \Sigma^{-1} (x - \mu)$. When completing the square, it is helpful to remember that for symmetric $\Sigma \succ 0$ and vectors x, μ ,

$$(x - \mu)^\top \Sigma^{-1} (x - \mu)$$

is always nonnegative and equals zero only when $x = \mu$.

References

- A. N. Kolmogorov. 1938. “On the Analytic Methods of Probability Theory.” *Rossiiskaya Akademiya Nauk*, no. 5: 5–41.
- Actor, Jonas. 2018. “Computation for the Kolmogorov Superposition Theorem.” {MS Thesis}, Rice.
- Albert, Jim. 1993. “A Statistical Analysis of Hitting Streaks in Baseball: Comment.” *Journal of the American Statistical Association* 88 (424): 1184–88.
- Altić, Mirela Slukan. 2013. “Exploring Along the Rome Meridian: Roger Boscovich and the First Modern Map of the Papal States.” In *History of Cartography: International Symposium of the ICA, 2012*, 71–89. Springer.
- Amazon. 2021. “The History of Amazon’s Forecasting Algorithm.”
- Amit, Yali, Gilles Blanchard, and Kenneth Wilder. 2000. “Multiple Randomized Classifiers: MRCL.”
- Andrews, D. F., and C. L. Mallows. 1974. “Scale Mixtures of Normal Distributions.” *Journal of the Royal Statistical Society. Series B (Methodological)* 36 (1): 99–102.
- Apley, Daniel W., and Jingyu Zhu. 2020. “Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models.” *Journal of the Royal Statistical Society Series B: Statistical Methodology* 82 (4): 1059–86.
- Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. “Wasserstein Generative Adversarial Networks.” *Proceedings of the 34th International Conference on Machine Learning*, 214–23.
- Armitage, Peter. 1975. *Sequential Medical Trials*. 2nd ed. Oxford: Blackwell Scientific Publications.
- Arnol’d, Vladimir I. 2006. “Forgotten and Neglected Theories of Poincaré.” *Russian Mathematical Surveys* 61 (1): 1.
- Ayala, Orlando, and Patrice Bechar. 2024. “Reducing Hallucination in Structured Outputs via Retrieval-Augmented Generation.” In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, 228–38. Mexico City, Mexico: Association for Computational Linguistics.
- Bach, Francis. 2024. “High-Dimensional Analysis of Double Descent for Linear Regression with Random Projections.” *SIAM Journal on Mathematics of Data Science* 6 (1): 26–50.
- Bachelier, Louis. 1900. “Théorie de La Spéculation.” PhD thesis, Paris: Université de Paris.

- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2014. “Neural Machine Translation by Jointly Learning to Align and Translate.” arXiv. <https://arxiv.org/abs/1409.0473>.
- Bai, Yuntao, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, et al. 2022. “Constitutional AI: Harmlessness from AI Feedback.” arXiv. <https://arxiv.org/abs/2212.08073>.
- Barron, Andrew R. 1993. “Universal Approximation Bounds for Superpositions of a Sigmoidal Function.” *IEEE Transactions on Information Theory* 39 (3): 930–45.
- Baum, Leonard E., Ted Petrie, George Soules, and Norman Weiss. 1970. “A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains.” *The Annals of Mathematical Statistics* 41 (1): 164–71.
- Behnia, Farnaz, Dominik Karbowski, and Vadim Sokolov. 2023. “Deep Generative Models for Vehicle Speed Trajectories.” *Applied Stochastic Models in Business and Industry* 39 (5): 701–19.
- Behrouz, Ali, and Mohammad Pezeshki. 2025. “MIRAS: Memory as an Optimization Object.” *Google Research*.
- Behrouz, Ali, Mohammad Pezeshki, and Rasool Fakoor. 2025. “Titans: Learning to Memorize at Test Time.” *arXiv Preprint arXiv:2501.00663*. <https://arxiv.org/abs/2501.00663>.
- Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. “Reconciling Modern Machine-Learning Practice and the Classical Bias–Variance Trade-Off.” *Proceedings of the National Academy of Sciences* 116 (32): 15849–54.
- Bellemare, Marc G., Will Dabney, and Rémi Munos. 2017. “A Distributional Perspective on Reinforcement Learning.” *Proceedings of the 34th International Conference on Machine Learning*, 449–58.
- Benda, Norbert, Michael Branson, Willi Maurer, and Tim Friede. 2016. “Sequential Designs with Small Samples: Evaluation and Recommendations for Normal Responses.” *Statistics in Medicine* 35 (19): 3215–30.
- Benoit, Dries F., and Dirk Van den Poel. 2012. “Binary Quantile Regression: A Bayesian Approach Based on the Asymmetric Laplace Distribution.” *Journal of Applied Econometrics* 27 (7): 1174–88.
- Berge, Travis, Nitish Sinha, and Michael Smolyansky. 2016. “Which Market Indicators Best Forecast Recessions?” *FEDS Notes*, August.
- Berry, Donald A. 1985. “Interim Analyses in Clinical Trials: Classical Vs. Bayesian Approaches.” *Statistics in Medicine* 4 (4): 521–26.
- Berry, Scott M., Bradley P. Carlin, J. Jack Lee, and Peter Müller. 2010. *Bayesian Adaptive Methods for Clinical Trials*. Boca Raton: CRC Press.
- Bhadra, Anindya, Jyotishka Datta, Nick Polson, Vadim Sokolov, and Jianeng Xu. 2021. “Merging Two Cultures: Deep and Statistical Learning.” *arXiv Preprint arXiv:2110.11561*. <https://arxiv.org/abs/2110.11561>.
- Billingsley, Patrick. 1995. *Probability and Measure*. 3rd ed. New York: John Wiley & Sons.

- Bird, Steven, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. Beijing ; Cambridge Mass.: O'Reilly Media.
- Bonfiglio, Rita, Annarita Granaglia, Raffaella Giocondo, Manuel Scimeca, and Elena Bonanno. 2021. "Molecular Aspects and Prognostic Significance of Microcalcifications in Human Pathology: A Narrative Review." *International Journal of Molecular Sciences* 22 (120).
- Bottou, Léon, Frank E Curtis, and Jorge Nocedal. 2018. "Optimization Methods for Large-Scale Machine Learning." *SIAM Review* 60 (2): 223–311.
- Braun, Heinrich, and Martin Riedmiller. 2009. "Constructive Neural Network Learning Algorithms for Pattern Classification." *IEEE Transactions on Neural Networks* 20 (1): 84–97.
- Breiman, Leo. 2001. "Random Forests." *Machine Learning* 45 (1): 5–32.
- Brier, Glenn W. 1950. "Verification of Forecasts Expressed in Terms of Probability." *Monthly Weather Review* 78 (1): 1–3.
- Brillinger, David R. 2012. "A Generalized Linear Model With 'Gaussian' Regressor Variables." In *Selected Works of David Brillinger*, edited by Peter Guttorp and David Brillinger, 589–606. Selected Works in Probability and Statistics. New York, NY: Springer.
- Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. 2020. "Language Models Are Few-Shot Learners." *Advances in Neural Information Processing Systems* 33: 1877–1901.
- Bryson, Arthur E. 1961. "A Gradient Method for Optimizing Multi-Stage Allocation Processes." In *Proc. Harvard Univ. Symposium on Digital Computers and Their Applications*. Vol. 72.
- Bryson, Arthur E., and Yu-Chi Ho. 1969. *Applied Optimal Control: Optimization, Estimation, and Control*. Waltham, MA: Blaisdell Publishing Company.
- Bumgarner, John M., Chad T. Lambert, Ayman A. Hussein, Daniel J. Cantillon, Bryan Baranowski, Kathy Wolski, Bruce D. Lindsay, Oussama M. Wazni, and Khaldoun G. Tarakji. 2018. "Smartwatch Algorithm for Automated Detection of Atrial Fibrillation." *Journal of the American College of Cardiology* 71 (21): 2381–88.
- Camerer, Colin F. 2003. *Behavioral Game Theory: Experiments in Strategic Interaction*. The Roundtable Series in Behavioral Economics. New York Princeton: Russell sage foundation Princeton university press.
- Campagnoli, Patrizia, Sonia Petrone, and Giovanni Petris. 2009. *Dynamic Linear Models with R*. New York, NY: Springer.
- Cannon, Alex J. 2018. "Non-Crossing Nonlinear Regression Quantiles by Monotone Composite Quantile Regression Neural Network, with Application to Rainfall Extremes." *Stochastic Environmental Research and Risk Assessment* 32 (11): 3207–25.
- Carlin, Bradley P., Nicholas G Polson, and David S Stoffer. 1992. "A Monte Carlo Approach to Nonnormal and Nonlinear State-Space Modeling." *Jour-*

- nal of the American Statistical Association* 87 (418): 493–500.
- Carter, Chris K, and Robert Kohn. 1994. “On Gibbs Sampling for State Space Models.” *Biometrika* 81 (3): 541–53.
- Carvalho, Carlos M, Hedibert F Lopes, Nicholas G Polson, and Matt A Taddy. 2010. “Particle Learning for General Mixtures.” *Bayesian Analysis* 5 (4): 709–40.
- Carvalho, Carlos M., Nicholas G. Polson, and James G. Scott. 2010. “The Horseshoe Estimator for Sparse Signals.” *Biometrika*, asq017.
- Chen, Charlie, Sébastien Borgeaud, Jean-Baptiste Alayrac, Eliza Buchatskaya, Sébastien Bodnariu, Benoit Steiner, Junteng Jia, et al. 2023. “Accelerating Large Language Model Decoding with Speculative Sampling.” *arXiv Preprint arXiv:2302.01318*. <https://arxiv.org/abs/2302.01318>.
- Chen, Cong, Naitee Li, Shuai Yuan, Zoran Antonijevic, Wei Guo, et al. 2022. “Application of Bayesian Methods to Accelerate Rare Disease Drug Development: Scopes and Hurdles.” *Orphanet Journal of Rare Diseases* 17: 186.
- Chen, Tianqi, and Carlos Guestrin. 2016. “XGBoost: A Scalable Tree Boosting System.” In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–94. New York, NY, USA: ACM.
- Chernozhukov, Victor, Iván Fernández-Val, and Alfred Galichon. 2010. “Quantile and Probability Curves Without Crossing.” *Econometrica : Journal of the Econometric Society* 78 (3): 1093–1125.
- Chib, Siddhartha. 1998. “Estimation and Comparison of Multiple Change-Point Models.” *Journal of Econometrics* 86 (2): 221–41.
- Choi, Hee Min, and James P Hobert. 2013. “Uniform Ergodicity of the Polya-Gamma Gibbs Sampler.” *Electronic Journal of Statistics* 7: 2054–64.
- Chroma Research. 2024. “Evaluating Chunking Strategies for Retrieval.”
- Chung, Hyung Won, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, et al. 2022. “Scaling Instruction-Finetuned Language Models.” arXiv. <https://arxiv.org/abs/2210.11416>.
- Clark, Kevin, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. “What Does BERT Look at? An Analysis of BERT’s Attention.” In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 276–86. Association for Computational Linguistics.
- Cootner, Paul H. 1967. *The Random Character of Stock Market Prices*. MIT press.
- Coppejans, Mark. 2004. “On Kolmogorov’s Representation of Functions of Several Variables by Functions of One Variable.” *Journal of Econometrics* 123 (1): 1–31.
- Cover, T., and P. Hart. 1967. “Nearest Neighbor Pattern Classification.” *IEEE Transactions on Information Theory* 13 (1): 21–27.
- Cover, Thomas M., and Joy A. Thomas. 2006. *Elements of Information The-*

- ory. John Wiley & Sons.
- Craven, Mark, and Jude W. Shavlik. 1996. “Extracting Tree-Structured Representations of Trained Networks.” In *Advances in Neural Information Processing Systems*, 8:24–30. MIT Press.
- Dabney, Will, Georg Ostrovski, David Silver, and Rémi Munos. 2018. “Implicit Quantile Networks for Distributional Reinforcement Learning.” arXiv. <https://arxiv.org/abs/1806.06923>.
- Dao, Tri. 2023. “FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning.” *arXiv Preprint arXiv:2307.08691*. <https://arxiv.org/abs/2307.08691>.
- Das, Payel, Subhajit Chaudhury, Elliot Nelson, Igor Melnyk, Sarath Swaminathan, Sihui Dai, Aurélie Lozano, et al. 2024. “Larimar: Large Language Models with Episodic Memory Control.” In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.
- Davison, Anthony Christopher. 2003. *Statistical Models*. Vol. 11. Cambridge university press.
- de Finetti, Bruno. 1937. “Foresight: Its Logical Laws, Its Subjective Sources.” In *Studies in Subjective Probability*, edited by Henry E. Kyburg and Howard E. Smokler, 93–158. New York: Wiley.
- . 1940. “Il Problema Dei Pieni.” *Giornale Dell’Istituto Italiano Degli Attuari* 11 (1): 1–88.
- Dean, Jeffrey, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, et al. 2012. “Large Scale Distributed Deep Networks.” In *Advances in Neural Information Processing Systems*, 1223–31.
- DeGroot, Morris H. 1974. “Reaching a Consensus.” *Journal of the American Statistical Association* 69 (345): 118–21.
- Dembo, Amir. 2021. “A Note on the Universal Approximation Capability of Deep Neural Networks.” *arXiv Preprint arXiv:2104.xxxxx*.
- DeMets, David L., and K. K. Gordon Lan. 1994. “Interim Analysis: The Alpha Spending Function Approach.” *Statistics in Medicine* 13 (13-14): 1341–52.
- Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–86. Minneapolis, Minnesota: Association for Computational Linguistics.
- Devroye, Luc. 1986. *Non-Uniform Random Variate Generation*. Springer Science & Business Media.
- Journal of the American Statistical Association 84 (408): 853–61.

Diaconis, Persi, and David Freedman. 1987. “A Dozen de Finetti-style Results in Search of a Theory.” In *Annales de l’IHP Probabilités Et Statistiques*, 23:397–423.

Diaconis, Persi, and Mehrdad Shahshahani. 1981. “Generating a Random Per-

- mutation with Random Transpositions.” *Probability Theory and Related Fields* 57 (2): 159–79.
- . 1984. “On Nonlinear Functions of Linear Combinations.” *SIAM Journal on Scientific and Statistical Computing* 5 (1): 175–91.
- Diaconis, P., and D. Ylvisaker. 1983. “Quantifying Prior Opinion.”
- Dietterich, Thomas G. 2000. “Ensemble Methods in Machine Learning.” In *Multiple Classifier Systems*, 1–15. Berlin, Heidelberg: Springer.
- Dixon, Mark J., and Stuart G. Coles. 1997. “Modelling Association Football Scores and Inefficiencies in the Football Betting Market.” *Journal of the Royal Statistical Society Series C: Applied Statistics* 46 (2): 265–80.
- Dixon, Matthew F, Nicholas G Polson, and Vadim O Sokolov. 2019. “Deep Learning for Spatio-Temporal Modeling: Dynamic Traffic Flows and High Frequency Trading.” *Applied Stochastic Models in Business and Industry* 35 (3): 788–807.
- Dreyfus, Stuart. 1962. “The Numerical Solution of Variational Problems.” *Journal of Mathematical Analysis and Applications* 5 (1): 30–45.
- . 1973. “The Computational Solution of Optimal Control Problems with Time Lag.” *IEEE Transactions on Automatic Control* 18 (4): 383–85.
- Duchi, John, Elad Hazan, and Yoram Singer. 2011. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.” *Journal of Machine Learning Research* 12 (61): 2121–59.
- Duflo, Esther, Michael Greenstone, Rohini Pande, and Nicholas Ryan. 2013. “Truth-Telling by Third-party Auditors and the Response of Polluting Firms: Experimental Evidence from India*.” *The Quarterly Journal of Economics* 128 (4): 1499–1545.
- Edwards, Ward, Harold Lindman, and Leonard J. Savage. 1963. “Bayesian Statistical Inference for Psychological Research.” *Psychological Review* 70 (3): 193–242.
- Efron, Bradley, and Carl Morris. 1975. “Data Analysis Using Stein’s Estimator and Its Generalizations.” *Journal of the American Statistical Association* 70 (350): 311–19.
- . 1977. “Stein’s Paradox in Statistics.” *Scientific American* 236 (5): 119–27.
- Enikolopov, Ruben, Vasily Korovkin, Maria Petrova, Konstantin Sonin, and Alexei Zakharov. 2013. “Field Experiment Estimate of Electoral Fraud in Russian Parliamentary Elections.” *Proceedings of the National Academy of Sciences* 110 (2): 448–52.
- Fawcett, Tom. 2006. “An Introduction to ROC Analysis.” *Pattern Recognition Letters* 27 (8): 861–74.
- Fedus, William, Barret Zoph, and Noam Shazeer. 2022. “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity.” *Journal of Machine Learning Research* 23 (120): 1–39.
- Fefferman, Charles L. 2006. “Existence and Smoothness of the Navier–Stokes Equation.” *The Millennium Prize Problems*, 57–67.
- Feller, William. 1971. *An Introduction to Probability Theory and Its Applications*.

- tions. Wiley.
- Feng, Guanhao, Nicholas G. Polson, and Jianeng Xu. 2016. “The Market for English Premier League (EPL) Odds.” *Journal of Quantitative Analysis in Sports* 12 (4). <https://arxiv.org/abs/1604.03614>.
- Fredholm, Ivar. 1903. “Sur Une Classe d’équations Fonctionnelles.” *Acta Mathematica* 27 (none): 365–90.
- Friedman, Jerome H. 2001. “Greedy Function Approximation: A Gradient Boosting Machine.” *Annals of Statistics*, 1189–1232.
- Friedman, Jerome H., and Werner Stuetzle. 1981. “Projection Pursuit Regression.” *Journal of the American Statistical Association* 76 (376): 817–23.
- Frühwirth-Schnatter, Sylvia, and Rudolf Frühwirth. 2007. “Auxiliary Mixture Sampling with Applications to Logistic Models.” *Computational Statistics & Data Analysis* 51 (April): 3509–28.
- . 2010. “Data Augmentation and MCMC for Binary and Multinomial Logit Models.” In *Statistical Modelling and Regression Structures: Festschrift in Honour of Ludwig Fahrmeir*, 111–32.
- Frühwirth-Schnatter, Sylvia, Rudolf Frühwirth, Leonhard Held, and Håvard Rue. 2008. “Improved Auxiliary Mixture Sampling for Hierarchical Models of Non-Gaussian Data.” *Statistics and Computing* 19 (4): 479.
- Frühwirth-Schnatter, Sylvia, and Helga Wagner. 2010. “Stochastic Model Specification Search for Gaussian and Partial Non-Gaussian State Space Models.” *Journal of Econometrics* 154: 85–100.
- Fukushima, Kunihiko. 1980. “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.” *Biological Cybernetics* 36 (4): 193–202.
- Gal, Yarin, and Zoubin Ghahramani. 2016. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning.” In *International Conference on Machine Learning*, 1050–59.
- Galton, Francis. 1907. “Vox Populi.” *Nature* 75: 450–51.
- Galushkin, A. I. 1973. “Synthesis of Multilayer Systems of Pattern Recognition.” *Neurocomputers and Their Application*.
- Gan, Link, and Alan Fritzler. 2016. “How to Become an Executive.”
- Gao, Luyu, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. “Precise Zero-Shot Dense Retrieval Without Relevance Labels.” *arXiv Preprint arXiv:2212.10496*. <https://arxiv.org/abs/2212.10496>.
- García-Arenzana, Nicolás, Eva María Navarrete-Muñoz, Virginia Lope, Pilar Moreo, Carmen Vidal, Soledad Laso-Pablos, Nieves Ascunce, et al. 2014. “Calorie Intake, Olive Oil Consumption and Mammographic Density Among Spanish Women.” *International Journal of Cancer* 134 (8): 1916–25.
- Gelman, Andrew, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. 2013. *Bayesian Data Analysis*. 3rd ed. Boca Raton: Chapman and Hall/CRC.
- Gleick, James. 1992. *Genius: The Life and Science of Richard Feynman*. New York: Pantheon Books.

- Gneiting, Tilmann, and Adrian E Raftery. 2007. “Strictly Proper Scoring Rules, Prediction, and Estimation.” *Journal of the American Statistical Association* 102 (477): 359–78.
- Goldstein, Alex, Adam Kapelner, Justin Bleich, and Emil Pitkin. 2015. “Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation.” *Journal of Computational and Graphical Statistics* 24 (1): 44–65.
- Gramacy, Robert B., and Nicholas G. Polson. 2012. “Simulation-Based Regularized Logistic Regression.” arXiv. <https://arxiv.org/abs/1005.3430>.
- Griewank, Andreas, Kshitij Kulshreshtha, and Andrea Walther. 2012. “On the Numerical Stability of Algorithmic Differentiation.” *Computing. Archives for Scientific Computing* 94 (2-4): 125–49.
- Guan, Xinyu, Li Lyra Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. “rStar-Math: Small LLMs Can Master Math Reasoning with Self-Evolved Deep Thinking.” arXiv. <https://arxiv.org/abs/2501.04519>.
- Gupte, Mihir, Eshan Dixit, Muhammad Tayyab, and Arun Adiththan. 2025. “What Works for ‘lost-in-the-Middle’ in LLMs? A Study on GM-extract and Mitigations.” *arXiv Preprint arXiv:2511.13900*. <https://arxiv.org/abs/2511.13900>.
- Hahn, P. Richard, Jared S. Murray, and Carlos M. Carvalho. 2020. “Bayesian Regression Tree Models for Causal Inference: Regularization, Confounding, and Heterogeneous Effects (with Discussion).” *Bayesian Analysis* 15 (3): 965–1056.
- Halevy, Alon, Peter Norvig, and Fernando Pereira. 2009. “The Unreasonable Effectiveness of Data.” *IEEE Intelligent Systems* 24 (2): 8–12.
- Hardt, Moritz, Ben Recht, and Yoram Singer. 2016. “Train Faster, Generalize Better: Stability of Stochastic Gradient Descent.” In *International Conference on Machine Learning*, 1225–34. PMLR.
- Hastie, Trevor, Andrea Montanari, Saharon Rosset, and Ryan J. Tibshirani. 2022. “Surprises in High-Dimensional Ridgeless Least Squares Interpolation.” *The Annals of Statistics* 50 (2): 949–86.
- Heath, David, and William Sudderth. 1976. “De Finetti’s Theorem on Exchangeable Variables.” *The American Statistician* 30 (4): 188–89.
- Heaton, J. B., and N. G. Polson. 2012. “Smart Money, Dumb Money: Learning Type from Price.” *Working Paper*.
- Heaton, J. B., N. G. Polson, and Jan Hendrik Witte. 2016. “Deep Learning for Finance: Deep Portfolios.” *Applied Stochastic Models in Business and Industry*.
- Held, Leonhard, and Chris C. Holmes. 2006. “Bayesian Auxiliary Variable Models for Binary and Multinomial Regression.” *Bayesian Analysis* 1 (1): 145–68.
- Hilgers, Ralf-Dieter, Kit Roes, and Nigel Stallard. 2016. “Efficient Ways Exist to Obtain the Optimal Sample Size in Clinical Trials in Rare Diseases.” *Journal of Clinical Epidemiology* 80: 68–76.

- Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky. 2012. “Neural Networks for Machine Learning, Lecture 6a: Overview of Mini-Batch Gradient Descent.” Coursera Lecture.
- Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. 2015. “Distilling the Knowledge in a Neural Network.” *arXiv Preprint arXiv:1503.02531*. <https://arxiv.org/abs/1503.02531>.
- Hou, Zhen, Hao Liu, Jiang Bian, Xing He, and Yan Zhuang. 2025. “Enhancing Medical Coding Efficiency Through Domain-Specific Fine-Tuned Large Language Models.” *Npj Health Systems* 2 (1): 14.
- Hsieh, Cheng-Yu, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. “Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes.” *arXiv Preprint arXiv:2305.02301*. <https://arxiv.org/abs/2305.02301>.
- Huang, Gao, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. 2017. “Snapshot Ensembles: Train 1, Get M for Free.” In *International Conference on Learning Representations*.
- Huber, Peter J. 1985. “Projection Pursuit.” *The Annals of Statistics* 13 (2): 435–75.
- Hyndman, Rob J., and George Athanasopoulos. 2021. *Forecasting: Principles and Practice*. 3rd ed. edition. Melbourne, Australia: Otexts.
- Igelnik, B., and N. Parikh. 2003. “Kolmogorov’s Spline Network.” *IEEE Transactions on Neural Networks* 14 (4): 725–33.
- Immer, Alexander, Matthias Bauer, Vincent Fortuin, Gunnar Rätsch, and Khan Mohammad Emtilay. 2021. “Scalable Marginal Likelihood Estimation for Model Selection in Deep Learning.” In *International Conference on Machine Learning*, 4563–73. PMLR.
- Irwin, Neil. 2016. “How to Become a C.E.O.? The Quickest Path Is a Winding One.” *The New York Times*, September.
- Jacobs, Robert A., Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. “Adaptive Mixtures of Local Experts.” *Neural Computation* 3 (1): 79–87.
- Jacquier, Eric, and Nicholas G. Polson. 2013. “Asset Allocation in Finance: A Bayesian Perspective.” In *Bayesian Theory and Applications*, edited by Paul Damien, Petros Dellaportas, Nicholas G. Polson, and David A. Stephens, 501–15. Oxford University Press.
- Januschowski, Tim, Yuyang Wang, Kari Torkkola, Timo Erkkilä, Hilaf Hasson, and Jan Gasthaus. 2022. “Forecasting with Trees.” *International Journal of Forecasting*, Special Issue: M5 competition, 38 (4): 1473–81.
- Jeffreys, Harold. 1998. *Theory of Probability*. Third Edition, Third Edition. Oxford Classic Texts in the Physical Sciences. Oxford, New York: Oxford University Press.
- Jiang, Huiqiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. “LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models.” *arXiv Preprint arXiv:2310.05736*. <https://arxiv.org/>

- [abs/2310.05736](https://arxiv.org/abs/2310.05736).
- Jiang, Wenxin, and Martin A. Tanner. 1999a. “Hierarchical Mixtures-of-Experts for Generalized Linear Models: Some Results on Denseness and Consistency.” In *Proceedings of the Sixteenth International Conference on Machine Learning*, 214–22. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
- . 1999b. “On the Identifiability of Mixtures-of-Experts.” *Neural Networks* 12 (9): 1253–58.
- Jiménez-Luna, José, Francesca Grisoni, Nils Weskamp, and Gisbert Schneider. 2020. “DrugEx V2: De Novo Design of Drug Molecule by Pareto-based Multi-Objective Reinforcement Learning in Polypharmacology.” *Journal of Cheminformatics* 12 (1): 1–12.
- Johannes, Michael S., Nick Polson, and Seung M. Yae. 2009. “Quantile Filtering and Learning.” *SSRN Electronic Journal*.
- Jumper, John, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, et al. 2021. “Highly Accurate Protein Structure Prediction with AlphaFold.” *Nature* 596 (7873): 583–89.
- Kaczmarz, Stefan. 1937. “Angenäherte Auflösung von Systemen Linearer Gleichungen.” *Bulletin International de l’Académie Polonaise Des Sciences Et Des Lettres* 35: 355–57.
- kaggle. 2020. “M5 Forecasting - Accuracy.”
- Kallenberg, Olav. 1997. *Foundations of Modern Probability*. 2nd ed. edition. Springer.
- Kalman, R. E., and R. S. Bucy. 1961. “New Results in Linear Filtering and Prediction Theory.” *Journal of Basic Engineering* 83 (1): 95–108.
- Kalman, Rudolph Emil. 1960. “A New Approach to Linear Filtering and Prediction Problems.” *Transactions of the ASME-Journal of Basic Engineering* 82 (Series D): 35–45.
- Kelly, J. L. 1956. “A New Interpretation of Information Rate.” *Bell System Technical Journal* 35 (4): 917–26.
- Keskar, Nitish Shirish, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.” *arXiv Preprint arXiv:1609.04836*. <https://arxiv.org/abs/1609.04836>.
- Keynes, John Maynard. 1930. “Economic Possibilities for Our Grandchildren.” In *Essays in Persuasion*, 358–73. W. W. Norton & Company.
- Khatab, Omar, and Matei Zaharia. 2020. “ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT.” In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 39–48. ACM.
- Kim, Young-Hoon, Jaehyung Shim, Hyoung-Seob Park, et al. 2024. “Diagnostic Accuracy of Single-Lead Handheld ECG Devices for Atrial Fibrillation Detection.” *Journal of Cardiovascular Electrophysiology* 35: 614–21.
- Kingma, Diederik, and Jimmy Ba. 2014. “Adam: A Method for Stochastic Optimization.” *ICML* 39: 348–56.

- timization.” *arXiv Preprint arXiv:1412.6980*. <https://arxiv.org/abs/1412.6980>.
- Klartag, Bo’az. 2007. “A Central Limit Theorem for Convex Sets.” *Inventiones Mathematicae* 168 (1): 91–131.
- Koenker, Roger. 2005. *Quantile Regression*. Econometric Society Monographs. Cambridge: Cambridge University Press.
- Kolmogoroff, Andrei. 1931. “Über Die Analytischen Methoden in Der Wahrscheinlichkeitsrechnung.” *Mathematische Annalen* 104 (1): 415–58.
- . 1933. *Grundbegriffe Der Wahrscheinlichkeitsrechnung*. Vol. 2. Ergebnisse Der Mathematik Und Ihrer Grenzgebiete. Berlin: Springer.
- Kolmogorov, AN. 1956. “On the Representation of Continuous Functions of Several Variables as Superpositions of Functions of Smaller Number of Variables.” In *Soviet. Math. Dokl*, 108:179–82.
- Kolmogorov, Andrey N. 1933. *Grundbegriffe Der Wahrscheinlichkeitsrechnung*. Berlin: Springer.
- Köppen, Mario. 2000. “The Curse of Dimensionality.” *5th Online World Conference on Soft Computing in Industrial Applications (WSC5)* 1: 4–8.
- LeCun, Yann, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2002. “Efficient Backprop.” In *Neural Networks: Tricks of the Trade*, 9–50. Springer.
- Leviathan, Yaniv, Matan Kalman, and Yossi Matias. 2023. “Fast Inference from Transformers via Predictive Sampling.” *arXiv Preprint arXiv:2211.17191*. <https://arxiv.org/abs/2211.17191>.
- Levina, Elizaveta, and Peter Bickel. 2001. “The Earth Mover’s Distance Is the Mallows Distance: Some Insights from Statistics.” In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, 2:251–56. IEEE.
- Lim, Bryan, Sercan Ö Arik, Nicolas Loeff, and Tomas Pfister. 2021. “Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting.” *International Journal of Forecasting* 37 (4): 1748–64.
- Lin, Zhouhan, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. “A Structured Self-attentive Sentence Embedding.” *arXiv*. <https://arxiv.org/abs/1703.03130>.
- Lindgren, Georg. 1978. “Markov Regime Models for Mixed Distributions and Switching Regressions.” *Scandinavian Journal of Statistics* 5 (2): 81–91.
- Lindley, D. V. 1961. “The Use of Prior Probability Distributions in Statistical Inference and Decisions.” In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, 4.1:453–69. University of California Press.
- Linnainmaa, Seppo. 1970. “The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors.” *Master’s Thesis (in Finnish), Univ. Helsinki*, 6–7.
- Liu, Hao, Matei Zaharia, and Pieter Abbeel. 2023. “Ring Attention with Blockwise Transformers for Near-Infinite Context.” *arXiv Preprint arXiv:2310.01889*. <https://arxiv.org/abs/2310.01889>.

- Liu, Nelson F., Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. “Lost in the Middle: How Language Models Use Long Contexts.” *Transactions of the Association for Computational Linguistics* 12: 157–73.
- Logunov, A. A. 2004. “Henri Poincare and Relativity Theory.” <https://arxiv.org/abs/physics/0408077>.
- Lorentz, George G. 1976. “The 13th Problem of Hilbert.” In *Proceedings of Symposia in Pure Mathematics*, 28:419–30. American Mathematical Society.
- Lundberg, Scott M, and Su-In Lee. 2017. “A Unified Approach to Interpreting Model Predictions.” In *Advances in Neural Information Processing Systems 30*, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, 4765–74. Curran Associates, Inc.
- Mackay, Charles. 1841. *Extraordinary Popular Delusions and the Madness of Crowds*. London: Richard Bentley.
- MacKay, David JC. 1992. “Bayesian Interpolation.” *Neural Computation* 4 (3): 415–47.
- Maharaj, Shiva, Nick Polson, and Vadim Sokolov. 2023. “Kramnik Vs Nakamura or Bayes Vs P-Value.” {{SSRN Scholarly Paper}}. Rochester, NY.
- Markowitz, Harry. 2006. “De Finetti Scoops Markowitz.” *Journal of Investment Management* 4 (3): 5.
- Mehrara, Nazanin, Yatao Zhong, Frederick Tung, Luke Bornn, and Greg Mori. 2017. “Learning Person Trajectory Representations for Team Activity Analysis.” *arXiv Preprint arXiv:1706.00893*. <https://arxiv.org/abs/1706.00893>.
- Metropolis, Nicholas. 1987. “The Beginning of the Monte Carlo Method.” *Los Alamos Science* 15: 125–30.
- Metropolis, Nicholas, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. 1953. “Equation of State Calculations by Fast Computing Machines.” *The Journal of Chemical Physics* 21 (6): 1087–92.
- Metropolis, Nicholas, and Stanislaw Ulam. 1949. “The Monte Carlo Method.” *Journal of the American Statistical Association* 44 (247): 335–41.
- Microsoft Research. 2024. “GraphRAG: Unlocking LLM Discovery on Narrative Private Data.”
- Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. “Efficient Estimation of Word Representations in Vector Space.” arXiv. <https://arxiv.org/abs/1301.3781>.
- Milman, Vitali D, and Gideon Schechtman. 2009. *Asymptotic Theory of Finite Dimensional Normed Spaces: Isoperimetric Inequalities in Riemannian Manifolds*. Vol. 1200. Springer.
- Minsky, Marvin, and Seymour Papert. 1969. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press.
- Miotto, Marilù, Nicola Rossberg, and Bennett Kleinberg. 2022. “Who Is GPT-3? An Exploration of Personality, Values and Demographics.” *arXiv*

- Preprint arXiv:2209.14338.* <https://arxiv.org/abs/2209.14338>.
- Mnih, Volodymyr, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2014. “Recurrent Models of Visual Attention.” *Advances in Neural Information Processing Systems* 27: 2204–12.
- Morris, Stephen. 1994. “Trade with Heterogeneous Prior Beliefs and No-Trade Theorems.” *Econometrica : Journal of the Econometric Society* 62 (6): 1327–47.
- . 1996. “Speculative Trade with Rational Beliefs.” *Journal of Economic Theory* 70 (2): 445–72.
- Nadaraya, E. A. 1964. “On Estimating Regression.” *Theory of Probability & Its Applications* 9 (1): 141–42.
- Nakkiran, Preetum, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2021. “Deep Double Descent: Where Bigger Models and More Data Hurt*.” *Journal of Statistical Mechanics: Theory and Experiment* 2021 (12): 124003.
- Nareklishvili, Maria, Nicholas Polson, and Vadim Sokolov. 2022. “Deep Partial Least Squares for Iv Regression.” *arXiv Preprint arXiv:2207.02612*. <https://arxiv.org/abs/2207.02612>.
- . 2023a. “Generative Causal Inference,” June. <https://arxiv.org/abs/2306.16096>.
- . 2023b. “Feature Selection for Personalized Policy Analysis,” July. <https://arxiv.org/abs/2301.00251>.
- Nelder, J. A., and R. W. M. Wedderburn. 1972. “Generalized Linear Models.” *Royal Statistical Society. Journal. Series A: General* 135 (3): 370–84.
- Nesterov, Yurii. 1983. “A Method of Solving a Convex Programming Problem with Convergence Rate $O(1/K^2)$.” In *Soviet Mathematics Doklady*, 27:372–76.
- Nicosia, Luca, Giulia Gnocchi, Ilaria Gorini, Massimo Venturini, Federico Fontana, Filippo Pesapane, Ida Abiuso, et al. 2023. “History of Mammography: Analysis of Breast Imaging Diagnostic Achievements over the Last Century.” *Healthcare* 11 (1596).
- Novick, Melvin R., and James E. Grizzle. 1965. “A Bayesian Approach to the Analysis of Data from Clinical Trials.” *Journal of the American Statistical Association* 60 (309): 81–96.
- Ostrovskii, GM, Yu M Volin, and WW Borisov. 1971. “Uber Die Berechnung von Ableitungen.” *Wissenschaftliche Zeitschrift Der Technischen Hochschule f Ur Chemie, Leuna-Merseburg* 13 (4): 382–84.
- Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, et al. 2022. “Training Language Models to Follow Instructions with Human Feedback.” *Advances in Neural Information Processing Systems* 35: 27730–44.
- Packer, Charles, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. 2023. “MemGPT: Towards LLMs as Operating Systems.” *arXiv Preprint arXiv:2310.08560*. <https://arxiv.org/abs/2310.08560>.

- Pan, Zhenyu, Haozheng Luo, Manling Li, and Han Liu. 2025. “Chain-of-Action: Faithful and Multimodal Question Answering Through Large Language Models.” *arXiv*. <https://arxiv.org/abs/2403.17359>.
- Park, Joon Sung, Lindsay Popowski, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2024. “Generative Agent Simulations of 1,000 People.” *arXiv Preprint arXiv:2411.10109*. <https://arxiv.org/abs/2411.10109>.
- Parzen, Emanuel. 2004. “Quantile Probability and Statistical Data Modeling.” *Statistical Science* 19 (4): 652–62.
- Petris, Giovanni. 2010. “An R Package for Dynamic Linear Models.” *Journal of Statistical Software* 36 (October): 1–16.
- Poincaré, Henri. 1898. “La Mesure Du Temps.” *Revue de Métaphysique Et de Morale* 6 (1): 1–13.
- . 1952. *Science and Hypothesis*. New York]: Dover Publications.
- Polson, Nicholas. 1996. “Convergence of Markov Chain Monte Carlo Algorithms (with Discussion).” *Bayesian Statistics* 5: 297–321.
- Polson, Nicholas G., and James G. Scott. 2012. “Good, Great, or Lucky? Screening for Firms with Sustained Superior Performance Using Heavy-Tailed Priors.” *The Annals of Applied Statistics* 6 (1): 161–85.
- . 2016. “Mixtures, Envelopes and Hierarchical Duality.” *Journal of the Royal Statistical Society Series B: Statistical Methodology* 78 (4): 701–27.
- Polson, Nicholas G., James G. Scott, and Jesse Windle. 2013. “Bayesian Inference for Logistic Models Using Pólya–Gamma Latent Variables.” *Journal of the American Statistical Association* 108 (504): 1339–49.
- Polson, Nicholas G., and Steven L. Scott. 2011. “Data Augmentation for Support Vector Machines.” *Bayesian Analysis* 6 (1): 1–23.
- Polson, Nicholas G., and James Scott. 2018. *AIQ: How People and Machines Are Smarter Together*. St. Martin’s Press.
- Polson, Nicholas G., and Vadim Sokolov. 2023. “Generative AI for Bayesian Computation.” <https://arxiv.org/abs/2305.14972>.
- Polson, Nicholas G., Vadim Sokolov, et al. 2017. “Deep Learning: A Bayesian Perspective.” *Bayesian Analysis* 12 (4): 1275–1304.
- Polson, Nicholas, and Vadim Sokolov. 2020. “Deep Learning: Computational Aspects.” *Wiley Interdisciplinary Reviews: Computational Statistics* 12 (5): e1500.
- Polson, Nicholas, Vadim Sokolov, and Jianeng Xu. 2021. “Deep Learning Partial Least Squares.” *arXiv Preprint arXiv:2106.14085*. <https://arxiv.org/abs/2106.14085>.
- Polson, Nick, Fabrizio Ruggeri, and Vadim Sokolov. 2024. “Generative Bayesian Computation for Maximum Expected Utility.” *Entropy. An International and Interdisciplinary Journal of Entropy and Information Studies* 26 (12): 1076.
- Polson, Nick, and Vadim Sokolov. 2025. “Negative Probability.” *Applied Stochastic Models in Business and Industry* 41 (1): e2910.
- Polson, Nick, Vadim Sokolov, and Jianeng Xu. 2023. “Quantum Bayesian

- Computation.” *Applied Stochastic Models in Business and Industry* 39 (6): 869–83.
- Polson, Nick, and Jan Hendrik Witte. 2014. “A Bellman View of Jesse Livermore.” arXiv. <https://arxiv.org/abs/1407.2642>.
- Qian, Chen, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, et al. 2024. “ChatDev: Communicative Agents for Software Development.” arXiv. <https://arxiv.org/abs/2307.07924>.
- Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. “Improving Language Understanding by Generative Pre-Training.” OpenAI.
- Rafailov, Rafael, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. “Direct Preference Optimization: Your Language Model Is Secretly a Reward Model.” arXiv. <https://arxiv.org/abs/2305.18290>.
- Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.” *Journal of Machine Learning Research* 21 (140): 1–67.
- Ramsey, Frank P. 1926. “Truth and Probability.” Histoy of {{Economic Thought Chapters}}. McMaster University Archive for the History of Economic Thought.
- Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. 2016. “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier.” In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–44. ACM.
- Riquelme, Carlos, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. 2021. “Scaling Vision with Sparse Mixture of Experts.” In *Advances in Neural Information Processing Systems*, 34:8583–95.
- Ritter, Hippolyt, Aleksandar Botev, and David Barber. 2018. “A Scalable Laplace Approximation For Neural Networks.”
- Robbins, Herbert, and Sutton Monro. 1951. “A Stochastic Approximation Method.” *The Annals of Mathematical Statistics* 22 (3): 400–407.
- Roberts, Gareth O., and Jeffrey S. Rosenthal. 2001. “Optimal Scaling for Various Metropolis-Hastings Algorithms.” *Statistical Science* 16 (4): 351–67.
- Rosenblatt, F. 1958. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” *Psychological Review* 65 (6): 386–408.
- Rubin, Hal S. Stern, John B. Carlin. 2015. *Bayesian Data Analysis*. 3rd ed. New York: Chapman and Hall/CRC.
- Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. 1986. “Learning Representations by Back-Propagating Errors.” *Nature* 323 (6088): 533.
- Salinas, David, Valentin Flunkert, and Jan Gasthaus. 2019. “DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks.”

- arXiv:1704.04110 [Cs, Stat]*, February. <https://arxiv.org/abs/1704.04110>.
- Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. “DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter.” *arXiv Preprint arXiv:1910.01108*. <https://arxiv.org/abs/1910.01108>.
- Sarthi, Parth, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. “RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval.” *arXiv Preprint arXiv:2401.18059*. <https://arxiv.org/abs/2401.18059>.
- Schmidhuber, Jürgen. 2015. “Deep Learning in Neural Networks: An Overview.” *Neural Networks* 61: 85–117.
- Schmidt-Hieber, Johannes. 2021. “The Kolmogorov–Arnold Representation Theorem Revisited.” *Neural Networks* 137 (May): 119–26.
- Schwertman, Neil C, AJ Gilks, and J Cameron. 1990. “A Simple Noncalculus Proof That the Median Minimizes the Sum of the Absolute Deviations.” *The American Statistician* 44 (1): 38–39.
- Scott, Steven L. 2002. “Bayesian Methods for Hidden Markov Models.” *Journal of the American Statistical Association* 97 (457): 337–51.
- . 2015. “Multi-Armed Bandit Experiments in the Online Service Economy.” *Applied Stochastic Models in Business and Industry* 31 (1): 37–45.
- Scott, Steven L. 2013. “Multi-Armed Bandit Experiments.”
- . 2022. “BoomSpikeSlab: MCMC for Spike and Slab Regression.”
- Scott, Steven L., and Hal R. Varian. 2015. “Bayesian Variable Selection for Nowcasting Economic Time Series.” In *Economic Analysis of the Digital Economy*, 119–35. University of Chicago Press.
- Scott, Steven, and Hal Varian. 2014. “Predicting the Present with Bayesian Structural Time Series.” *Int. J. Of Mathematical Modelling and Numerical Optimisation* 5 (January): 4–23.
- Seidenfeld, Tedd. 1984. “Comments on Causal Decision Theory.” *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association* 1984 (2): 201–12.
- Sellke, Thomas, M. J Bayarri, and James O Berger. 2001. “Calibration of ρ Values for Testing Precise Null Hypotheses.” *The American Statistician* 55 (1): 62–71.
- Selvaraju, Ramprasaath R., Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.” In *Proceedings of the IEEE International Conference on Computer Vision*, 618–26. IEEE.
- Shapiro, Sam. 1988. “Selection, Follow-up, and Analysis in the Health Insurance Plan Study: A Randomized Trial with Breast Cancer Screening.” *Journal of the National Cancer Institute* 80 (14): 1125–32.
- Shazeer, Noam, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.” In *International*

- Conference on Learning Representations.*
- Shimony, Abner. 1955. “Coherence and the Axioms of Confirmation.” *The Journal of Symbolic Logic* 20 (1): 1–28.
- Shinn, Noah, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. “Reflexion: Language Agents with Verbal Reinforcement Learning.” <https://arxiv.org/abs/2303.11366>.
- Shiryayev, A. N. 1992. “On Analytical Methods in Probability Theory.” In *Selected Works of a. N. Kolmogorov: Volume II Probability Theory and Mathematical Statistics*, edited by A. N. Shiryayev, 62–108. Dordrecht: Springer Netherlands.
- Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. 2013. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.” *arXiv Preprint arXiv:1312.6034*. <https://arxiv.org/abs/1312.6034>.
- Simpson, Edward. 2010. “Edward Simpson: Bayes at Bletchley Park.” *Significance* 7 (2): 76–80.
- Singh, Pratyush Kumar, Kathryn A. Farrell-Maupin, and Danial Faghihi. 2024. “A Framework for Strategic Discovery of Credible Neural Network Surrogate Models Under Uncertainty.” *arXiv*. <https://arxiv.org/abs/2403.08901>.
- Smith, A. F. M. 1975. “A Bayesian Approach to Inference about a Change-Point in a Sequence of Random Variables.” *Biometrika* 62 (2): 407–16.
- Sokolov, Vadim. 2017. “Discussion of ‘Deep Learning for Finance: Deep Portfolios?’” *Applied Stochastic Models in Business and Industry* 33 (1): 16–18.
- Spiegelhalter, David, and Yin-Lam Ng. 2009. “One Match to Go!” *Significance* 6 (4): 151–53.
- Sprecher, David A. 1965. “On the Structure of Continuous Functions of Several Variables.” *Transactions of the American Mathematical Society* 115: 340–55.
- Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” *Journal of Machine Learning Research* 15 (1): 1929–58.
- Stern, Hal S. 1994. “A Brownian Motion Model for the Progress of Sports Scores.” *Journal of the American Statistical Association* 89 (427): 1128–34.
- Stern, H., Adam Sugano, J. Albert, and R. Koning. 2007. “Inference about Batter-Pitcher Matchups in Baseball from Small Samples.” *Statistical Thinking in Sports*, 153–65.
- Stigler, Stephen M. 1981. “Gauss and the Invention of Least Squares.” *The Annals of Statistics*, 465–74.
- Stroud, Jonathan R., Peter Müller, and Nicholas G. Polson. 2003. “Nonlinear State-Space Models with State-Dependent Variances.” *Journal of the American Statistical Association* 98 (462): 377–86.
- Sundararajan, Mukund, Ankur Taly, and Qiqi Yan. 2017. “Axiomatic Attri-

- bution for Deep Networks.” In *Proceedings of the 34th International Conference on Machine Learning*, 3319–28. PMLR.
- Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton. 2013. “On the Importance of Initialization and Momentum in Deep Learning.” In *International Conference on Machine Learning*, 1139–47.
- Tabar, Laszlo, CJ Gad Fagerberg, Anders Gad, Lennart Balderup, Lars H Holmberg, Ove Grøntoft, Ulf Ljungquist, et al. 1985. “Reduction in Mortality from Breast Cancer After Mass Screening with Mammography: Randomised Trial from the Breast Cancer Screening Working Group of the Swedish National Board of Health and Welfare.” *The Lancet* 325 (8433): 829–32.
- Taleb, Nassim Nicholas. 2007. *The Black Swan: The Impact of the Highly Improbable*. Annotated edition. New York. N.Y: Random House.
- Thompson, William R. 1933. “On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples.” *Biometrika* 25 (3/4): 285–94.
- Tiao, Louis. 2019. “Pólya-Gamma Bayesian Logistic Regression.” Blog post.
- Tikhonov, Andrei N. 1963. “Solution of Incorrectly Formulated Problems and the Regularization Method.” *Sov Dok* 4: 1035–38.
- Tikhonov, Andrey Nikolayevich et al. 1943. “On the Stability of Inverse Problems.” In *Dokl. Akad. Nauk Sssr*, 39:195–98.
- Tsai, Yao-Hung Hubert, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2019. “Transformer Dissection: A Unified Understanding of Transformer’s Attention via the Lens of Kernel.” arXiv. <https://arxiv.org/abs/1908.11775>.
- Turing, A. M. 1950. “Computing Machinery and Intelligence.” *Mind; a Quarterly Review of Psychology and Philosophy* 59 (236): 433–60.
- U.S. Food and Drug Administration. 2010. “Guidance for the Use of Bayesian Statistics in Medical Device Clinical Trials.”
- Varian, Hal R. 2010. “Computer Mediated Transactions.” *American Economic Review* 100 (2): 1–10.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2023. “Attention Is All You Need.” arXiv. <https://arxiv.org/abs/1706.03762>.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. “Attention Is All You Need.” *Advances in Neural Information Processing Systems* 30: 5998–6008.
- Veblen, Thorstein. 1899. *The Theory of the Leisure Class: An Economic Study of Institutions*. New York: Macmillan.
- . 1921. *The Engineers and the Price System*. New York: B. W. Huebsch.
- Vecer, Jan, Frantisek Kopriva, and Tomoyuki Ichiba. 2009. “Estimating the Effect of the Red Card in Soccer: When to Commit an Offense in Exchange for Preventing a Goal Opportunity.” *Journal of Quantitative Analysis in Sports* 5 (1).

- Villar, Sofía S., Jack Bowden, and James Wason. 2015. “Multi-Armed Bandit Models for the Optimal Design of Clinical Trials: Benefits and Challenges.” *Statistical Science* 30 (2): 199–215.
- Ville, Jean. 1939. “Étude Critique de La Notion de Collectif.” Thèses de l’entre-Deux-Guerres. PhD thesis, Université de Paris.
- Viterbi, A. 1967. “Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm.” *IEEE Transactions on Information Theory* 13 (2): 260–69.
- Wachter, Sandra, Brent Mittelstadt, and Chris Russell. 2017. “Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR.” *Harvard Journal of Law & Technology* 31: 841–87.
- Wald, Abraham. 1945. “Sequential Tests of Statistical Hypotheses.” *The Annals of Mathematical Statistics* 16 (2): 117–86.
- . 1947. *Sequential Analysis*. New York: John Wiley & Sons.
- Wang, Shaun. 1996. “Premium Calculation by Transforming the Layer Premium Density.” *ASTIN Bulletin* 26 (1): 71–92.
- Watanabe, Sumio. 2013. “A Widely Applicable Bayesian Information Criterion.” *The Journal of Machine Learning Research* 14 (1): 867–97.
- Watson, Geoffrey S. 1964. “Smooth Regression Analysis.” *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 26 (4): 359–72.
- Wei, Bo, Thomas M. Braun, Roy N. Tamura, and Kelley Kidwell. 2018. “A Small n Sequential Multiple Assignment Randomized Trial Design for Use in Rare Disease Research.” *Statistics in Medicine* 37 (26): 3836–52.
- Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.” arXiv. <https://arxiv.org/abs/2201.11903>.
- Werbos, Paul. 1974. “Beyond Regression:” New Tools for Prediction and Analysis in the Behavioral Sciences.” *Ph. D. Dissertation, Harvard University*.
- Werbos, Paul J. 1982. “Applications of Advances in Nonlinear Sensitivity Analysis.” In *System Modeling and Optimization*, 762–70. Springer.
- West, Mike, and Jeff Harrison. 1997. *Bayesian Forecasting and Dynamic Models*. Springer.
- Wiener, Norbert. 1950. *The Human Use of Human Beings: Cybernetics and Society*. Boston: Houghton Mifflin.
- Windle, Jesse, Nicholas G. Polson, and James G. Scott. 2014. “Sampling Polyalpha-Gamma Random Variates: Alternate and Approximate Techniques.” arXiv. <https://arxiv.org/abs/1405.0506>.
- Yaari, Menahem E. 1987. “The Dual Theory of Choice Under Risk.” *Econometrica : Journal of the Econometric Society* 55 (1): 95–115.
- Yang, An, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, et al. 2025. “Qwen2. 5-1m Technical Report.” *arXiv Preprint arXiv:2501.15383*. <https://arxiv.org/abs/2501.15383>.
- Yao, Shunyu, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. “Tree of Thoughts: Deliberate Prob-

- lem Solving with Large Language Models.” arXiv. <https://arxiv.org/abs/2305.10601>.
- Ye, Tong, Shijing Si, Jianzong Wang, Ning Cheng, Zhitao Li, and Jing Xiao. 2023. “On the Calibration and Uncertainty with Pólya-Gamma Augmentation for Dialog Retrieval Models.” In *Proceedings of the AAAI Conference on Artificial Intelligence*. <https://arxiv.org/abs/2303.08606>.
- Ye, Yixin, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. “LIMO: Less Is More for Reasoning.” arXiv. <https://arxiv.org/abs/2502.03387>.
- Yu, Gyeong-In, Joo Seong Jeong, Geon-Woo Kim, Soo-Jin Jeong, Woosung Lee, and Byung-Gon Chun. 2022. “Orca: A Distributed Serving System for Transformer-based Generative Models.” In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 527–46.
- Zhang, Wei, Katsuyuki Itoh, Jun Tanida, and Yoshiki Ichioka. 1988. “Shift-Invariant Pattern Recognition Neural Network and Its Optical Architecture.” *Proceedings of Annual Conference of the Japan Society of Applied Physics*.
- Zhang, Yichi, Anirban Datta, and Sudipto Banerjee. 2018. “Scalable Gaussian Process Classification with Pólya-Gamma Data Augmentation.” *arXiv Preprint arXiv:1802.06383*. <https://arxiv.org/abs/1802.06383>.