Lecture 3:

From Set to Linear Regression

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com

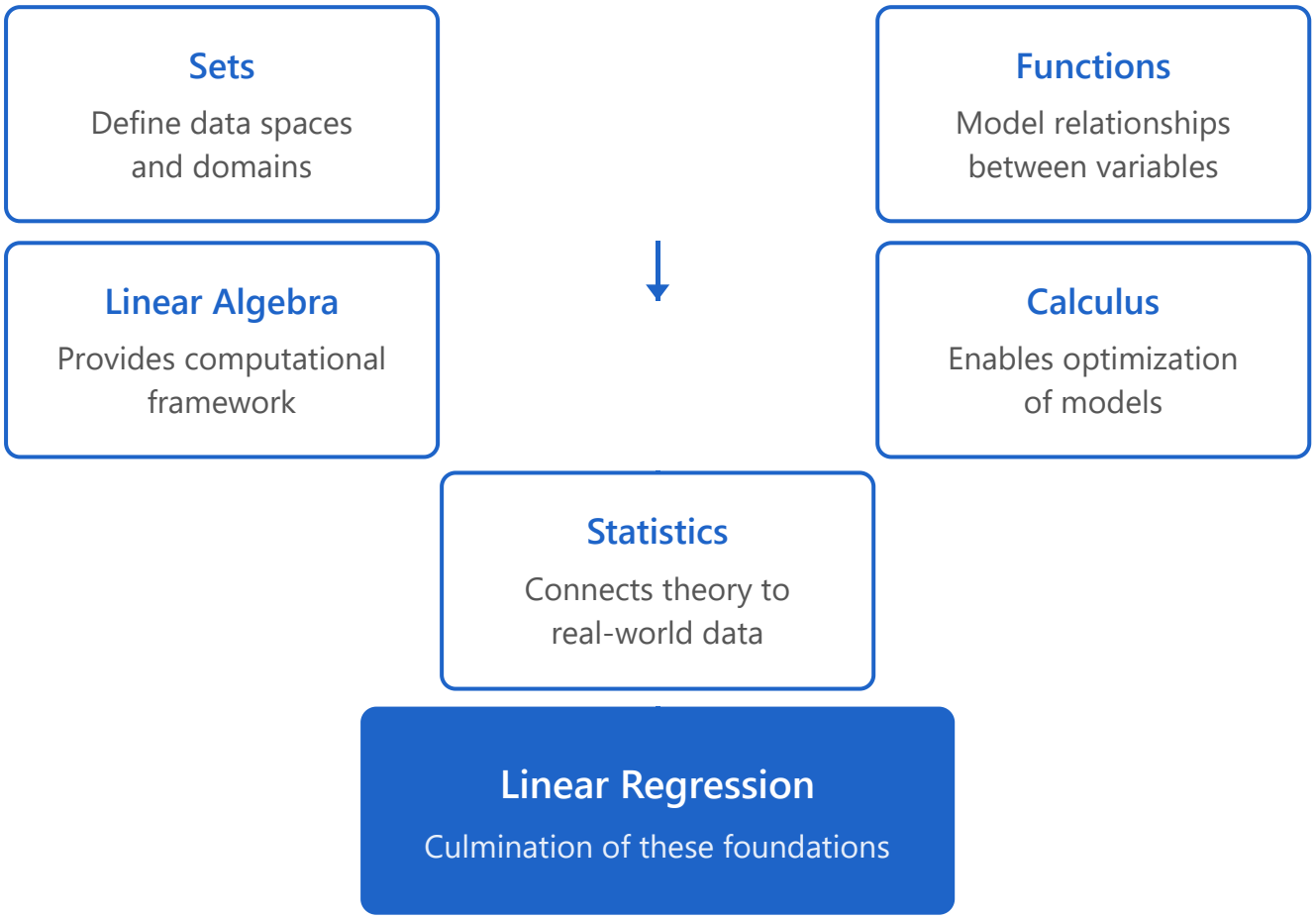
Lecture Contents

Part 1/3:

Mathematical Foundations

1. Course Overview - From Sets to Regression
2. Set Theory Basics and Notation
3. Functions and Mapping Concepts
4. Vector Spaces and Basis
5. Inner Product and Orthogonality
6. Matrix Operations and Properties
7. Inverse Matrices and Determinants
8. Eigenvalues and Eigenvectors
9. Differentiation and Partial Derivatives

Journey from Abstract Mathematics to Practical Machine Learning



Goal: Build intuition for why math matters in ML

Set Theory Basics and Notation

Set: Collection of distinct objects
$X = \{x_1, x_2, ..., x_n\}$

Common sets: $\mathbb{R}$ (real numbers)
$\mathbb{R}^n$ (n-dimensional space)

Membership: $x \in X$ means "x belongs to set X"

Subset: $A \subseteq B$ means all elements of A are in B

Operations: Union ($\cup$), Intersection ($\cap$)
Complement (')

Cartesian product: $X \times Y = \{(x,y) \mid x \in X, y \in Y\}$

Set Operations

$A \cup B$ Union

$A \cap B$ Intersection

ML Application

Data points live in feature space $\mathbb{R}^n$

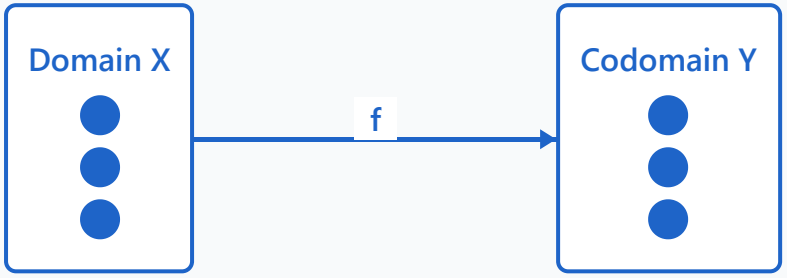
Example: Image data → $\mathbb{R}^{784}$ (28×28 pixels)

Sets define **domains** for regression functions

Sets define domains for our regression functions

Functions and Mapping Concepts

Function: f: X → Y



Each input maps to **exactly one** output

Domain X: Set of all possible inputs

Codomain Y: Set of all possible outputs

Range: Actual outputs achieved by f

One-to-one (Injective)

Different inputs → Different outputs
No two inputs map to the same output

Onto (Surjective)

Every output is reached
Range = Codomain

Bijection

Both injective and surjective
Perfect one-to-one correspondence

ML Application

Regression models are functions
f(x) = y

Function Mapping Examples

Natural Number Division

f: ℕ × ℕ\{0} → ℚ

- f(6, 2) = 3
- f(7, 2) = 3.5

Square Function

f: ℝ → ℝ≥0

- f(-3) = 9
- f(0) = 0

Absolute Value

f: ℝ → ℝ≥0

- f(-5) = 5
- f(0) = 0

• f(10, 4) = 2.5

✓ Well-defined function
Each pair maps to unique rational

• f(4) = 16

✓ Surjective onto non-negative reals
✗ Not injective (±x → same output)

• f(3) = 3

✓ Surjective onto non-negative reals
✗ Not injective (±x → same output)

Vector Spaces and Basis

Vector Space V

Set closed under **addition** and **scalar multiplication**

- $\mathbb{R}^n$ (n-dimensional space)
- Polynomial space
- Function space

Linear Combination

$v = c_1v_1 + c_2v_2 + \ldots + c_nv_n$

Span

All possible linear combinations of vectors

Linear Independence

No vector is combination of others

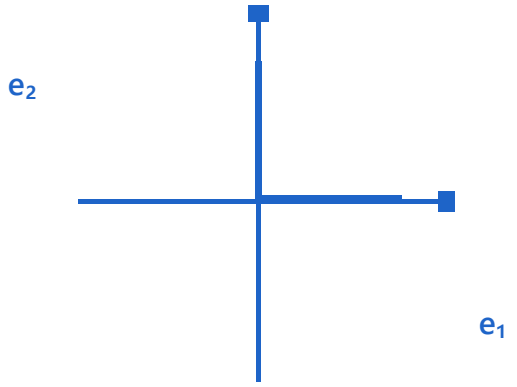
Basis

Minimal spanning set (linearly independent)

Dimension

Number of basis vectors

Standard Basis in $\mathbb{R}^2$



Standard Basis in $\mathbb{R}^n$

$$e_1 = (1, 0, \ldots, 0)$$

$$e_2 = (0, 1, \ldots, 0)$$

Any vector $v = c_1e_1 + c_2e_2 + \ldots + c_ne_n$

Inner Product and Orthogonality

Inner Product (Dot Product)

$$\langle x, y \rangle = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n$$

Geometric Interpretation

$$\langle x, y \rangle = \|x\| \|y\| \cos(\theta)$$

Measures alignment between vectors

Norm (Length)

$$\|x\| = \sqrt{(\langle x, x \rangle)} = \sqrt{(x_1^2 + x_2^2 + \ldots + x_n^2)}$$

Orthogonality

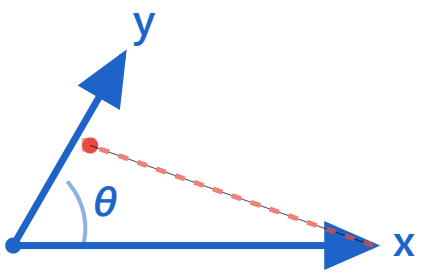
$$x \perp y \text{ when } \langle x, y \rangle = 0$$

Vectors are perpendicular ($\theta = 90°$)

Orthonormal Basis

Basis vectors with unit length, mutually orthogonal

Geometric Interpretation



Inner product ⟨x, y⟩ measures how much x "projects onto" y

Dashed line shows the projection of x onto y

Key for Regression

Projection

$$\text{proj}_y(x) = (\langle x, y\rangle / \langle y, y\rangle) \cdot y$$

Component of x in direction of y

- Projecting data onto subspaces
- Residuals are orthogonal to fitted values
- Minimizing distance = maximizing projection

Matrix Operations and Properties

Matrix $A \in \mathbb{R}^{m \times n}$

Rectangular array of numbers

$$\begin{bmatrix} a_{1\,1} & a_{1\,2} & \cdots & a_{1\,n} \\ a_{2\,1} & a_{2\,2} & \cdots & a_{2\,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Matrix-Vector Multiplication

Ax represents linear transformation

Matrix Multiplication

$(AB)_{ij} = \Sigma_k A_{ik} B_{kj}$

Transpose

A^T swaps rows and columns

Symmetric Matrix

$A = A^T$ (important in regression)

Identity Matrix

Matrix Multiplication Example

$$\underset{m \times k}{A} \times \underset{k \times n}{B} = \underset{m \times n}{AB}$$

Identity Matrix I

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Key Properties

$$(AB)^T = B^T A^T$$

$$(AB)C = A(BC)$$

ML Application

Matrices encode systems of linear equations

$AI = IA = A$

Lecture 4: From Linear to Logistic Regression ## 📋 Overview **Instructor:** Ho-min Park **Email:** homin.park@ghent.ac.kr | powersimmani@gmail.com **Total Slides:** 31 **Lecture Duration:** Approximately 150-180 minutes (3 parts) **Difficulty Level:** Intermediate **Course Level:** Undergraduate/Graduate Introduction to Machine Learning This lecture bridges the gap between regression and classification problems, starting with advanced linear regression techniques and culminating in logistic regression for binary and multiclass classification. The material progresses from polynomial regression and regularization methods through linear classifiers to the complete derivation and implementation of logistic regression. --- ## 🎯 Learning Objectives By the end of this lecture, students will be able to: 1. **Apply advanced linear regression techniques** including polynomial regression, Ridge (L2), Lasso (L1), and Elastic Net regularization to handle overfitting and feature selection 2. **Understand the fundamental differences** between regression and classification problems, and explain why linear regression fails for classification tasks 3. **Derive and implement logistic regression** from first principles using maximum likelihood estimation and gradient descent optimization 4. **Extend binary classification** to multiclass problems using both One-vs-Rest strategy and Softmax regression 5. **Evaluate and deploy** logistic regression models with appropriate regularization, handling class imbalance, and selecting proper evaluation metrics --- ## 🧮 Lecture Structure ### Part 1/3: Advanced Linear Regression (Slides 3-11) **Duration:** 50-60 minutes #### Topics Covered: 1. **Review and Connection to Previous Lecture** - Linear regression minimizes sum of squared errors - Normal equation vs. gradient descent solutions - Core assumptions: linearity, independence, homoscedasticity - Challenges: overfitting, multicollinearity, noise sensitivity 2. **Revisiting Linear Regression Assumptions** - Linearity: relationship between X and y is linear - Independence: observations are independent - Homoscedasticity: constant variance of residuals - Normality: residuals follow normal distribution - No multicollinearity: predictors not highly correlated - Impact of assumption violations on prediction accuracy 3. **Polynomial Regression and Basis Expansion** - Transform features: $x \rightarrow x, x^2, x^3, \ldots$ - Example equation: $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$ - Still "linear" in parameters (coefficients) - Trade-off between model flexibility and overfitting risk - Validation approach for choosing polynomial degree - Other basis functions: logarithmic, exponential, trigonometric - Interactive Desmos simulator included 4. **Ridge Regression (L2 Regularization)** - Cost function: RSS + $\lambda\Sigma\beta_i^2$ - Shrinks coefficients towards zero (but never exactly zero) - Handles multicollinearity effectively - Stabilizes predictions with correlated features - $\lambda = 0 \rightarrow$ standard linear regression; $\lambda \rightarrow \infty \rightarrow$ all coefficients $\rightarrow 0$ - Hyperparameter tuning via cross-validation 5. **Lasso Regression (L1 Regularization)** - Cost function: RSS + $\lambda\Sigma|\beta_i|$ - Can shrink coefficients to exactly zero (automatic feature selection) - Produces sparse models with fewer features - Useful for high-dimensional data - Interpretability advantage through feature elimination 6. **Elastic Net** - Cost function: RSS + $\lambda_1\Sigma\beta_i^2 + \lambda_2\Sigma|\beta_i|$ - Combines benefits of both Ridge and Lasso - Two hyperparameters: α (mixing ratio) and λ (strength) - $\alpha = 0 \rightarrow$ Ridge; $0 < \alpha < 1 \rightarrow$ Elastic Net; $\alpha = 1 \rightarrow$ Lasso - Particularly useful when predictors are highly correlated 7. **Feature Selection and Importance** - Lasso's automatic feature selection mechanism - Coefficient magnitude interpretation - Regularization path visualization - Trade-off between model complexity and performance 8. **Limitations of Linear Regression** - Cannot handle non-linear decision boundaries naturally - Assumes continuous output (not suitable for classification) - Sensitive to outliers without robust variants - Feature engineering required for complex patterns - Interpretation difficult with many interaction terms - Limited for categorical outcomes #### Key Concepts: - **Regularization:** Technique to prevent overfitting by adding penalty terms to the loss function - **Bias-Variance Trade-off:** Balance between model complexity and generalization - **Feature Engineering:** Transforming input features to capture non-linear relationships - **Sparsity:** Property of having many zero coefficients, leading to simpler models - **Hyperparameter Tuning:** Process of selecting optimal regularization strength #### Learning Outcomes: - Students can identify when to apply polynomial regression vs. regularization - Students can explain the mathematical differences between L1 and L2 penalties - Students can implement Ridge, Lasso, and Elastic Net using scikit-learn - Students can interpret feature importance from regularized models - Students understand the limitations that motivate classification algorithms --- ### Part 2/3: Transition to Classification (Slides 12-20) **Duration:** 50-60 minutes #### Topics Covered: 9. **Regression vs Classification Problems** - Regression: predict continuous values (price, temperature, age) - Classification: predict discrete categories (spam/ham, disease/healthy) - Key difference: output space $\mathbb{R}$ (continuous) vs. finite set (discrete) - Regression output: $y \in \mathbb{R}$ (e.g., $\hat{y} = 25.7°C$) - Classification output: $y \in \{0, 1, \ldots, K-$

1} (e.g., $\hat{y} = 1$ for Spam) - Binary classification focus: two classes (0 and 1) 10. **Linear Classifier Concepts** - Goal: find a line (hyperplane) that separates two classes - Decision function: $f(x) = w^T x + b$ - Classification rule: if $f(x) > 0 \rightarrow$ Class 1; else $\rightarrow$ Class 0 - Geometric view: distance from decision boundary - Weight vector w: perpendicular to decision boundary - Bias term b: shifts boundary position - Extension to high-dimensional space (hyperplanes) 11. **Perceptron Algorithm** - Simplest linear classifier (Frank Rosenblatt, 1957) - Update rule: $w \leftarrow w + \eta(y - \hat{y})x$ - Activation function: step function (0 or 1) - Iteratively adjusts weights for misclassified points - Converges if data is linearly separable - Limitation: no convergence for non-separable data - Historical significance: foundation of neural networks 12. **Decision Boundaries and Linear Separability** - Definition of linear separability - Hyperplane equation in n-dimensional space - Examples of linearly separable vs. non-separable datasets - XOR problem as classic non-separable case - Geometric interpretation of margins 13. **Why Linear Regression Fails for Classification** - Problem 1: Predictions can be < 0 or > 1 - Problem 2: Treats classes as ordered numerical values - Problem 3: Sensitive to outliers in feature space - Problem 4: Squared loss inappropriate for binary outcomes - Example: predicting disease probability (should be in [0,1]) - Solution requirement: output bounded to [0,1] representing probability 14. **Odds and Log Odds** - Probability: $P(\text{event}) \in [0, 1]$ - Odds: $p / (1 - p) \in [0, \infty)$ - Log odds (Logit): $\log(p / (1 - p)) \in (-\infty, \infty)$ - Key insight: log odds can take any real value - Example: $P = 0.8 \rightarrow$ Odds $= 4 \rightarrow$ Log odds $= 1.39$ - Bridge to logistic regression formulation 15. **Introduction to Sigmoid Function** - Definition: $\sigma(z) = 1 / (1 + e^{-z})$ - Maps any real number to (0,1) range - Perfect for probability interpretation - S-shaped curve: smooth transition between 0 and 1 - Solves the "prediction outside [0,1]" problem - Output: $P(y=1|x) = \sigma(w^T x + b)$ - Decision threshold: $z=0 \rightarrow \sigma=0.5$ 16. **Properties of Logistic Function** - Symmetry: $\sigma(z) + \sigma(-z) = 1$ - Monotonic: always increasing, never decreases - Smooth: differentiable everywhere (no jumps) - Bounded: output always in (0,1), never exactly 0 or 1 - Interpretable: steepness indicates confidence - Derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ (useful for gradient descent) - Comparison to step function: smooth vs. hard threshold #### Key Concepts: - **Linear Separability:** Data can be perfectly separated by a hyperplane - **Decision Boundary:** The line/hyperplane that separates different classes - **Sigmoid Function:** Mathematical function that maps real numbers to probabilities - **Logit Transform:** Converting probabilities to log-odds for linear modeling - **Probability Interpretation:** Output as conditional probability $P(y=1|x)$ #### Learning Outcomes: - Students can distinguish between regression and classification tasks - Students understand why linear regression is inappropriate for classification - Students can derive the sigmoid function as a solution to classification constraints - Students can explain the perceptron algorithm and its limitations - Students grasp the mathematical foundation for logistic regression --- ### Part 3/3: Completing Logistic Regression (Slides 21-30) **Duration:** 50-60 minutes #### Topics Covered: 17. **Defining the Logistic Regression Model** - Complete model specification - Linear combination: $z = w^T x + b$ - Probability output: $P(y=1|x) = \sigma(w^T x + b)$ - Decision rule: predict class 1 if $P(y=1|x) \geq 0.5$ - Model assumptions and interpretations - Relationship to generalized linear models 18. **Maximum Likelihood Estimation (MLE)** - Goal: find parameters that maximize likelihood of observed data - Likelihood function: $L(w,b) = \prod P(y_i|x_i)$ - Log-likelihood (easier to optimize): $\ell = \Sigma \log P(y_i|x_i)$ - For binary classification: $\ell = \Sigma[y_i \log(p) + (1-y_i)\log(1-p)]$ - Maximizing log-likelihood = minimizing negative log-likelihood - Why MLE? Principled probabilistic approach - Connection to cross-entropy loss function 19. **Binary Cross-Entropy Loss** - Loss function: $L = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$ - Penalizes confident wrong predictions heavily - Equivalent to negative log-likelihood for Bernoulli distribution - Convex function (has single global minimum) - Average loss over dataset: $BCE = -(1/n)\Sigma[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$ - Gradient: $\partial L/\partial w = (\hat{y} - y)x$ (elegant and simple!) 20. **Applying Gradient Descent** - Iterative optimization algorithm - Update rule: $w \leftarrow w - \eta \nabla L(w)$ - For logistic regression: $w \leftarrow w - \eta(\hat{y} - y)x$ - Learning rate η controls step size - Convergence criteria: change in loss < threshold - Batch vs. stochastic vs. mini-batch gradient descent - Practical considerations: learning rate scheduling, momentum 21. **Multiclass - One-vs-Rest Strategy** - Extension to $K > 2$ classes - Train K binary classifiers: Class k vs. all others - For each class: separate logistic regression model - Prediction: choose class with highest probability - Advantages: simple, interpretable, parallelizable - Disadvantage: probabilities may not sum to 1 - Works well when classes are well-separated 22. **Softmax Regression** - Natural extension to multiclass classification - Softmax function: $P(y=k|x) = \exp(z_k) / \Sigma_j \exp(z_j)$ - Properties: outputs sum to 1, differentiable, interpretable - Each class has its own weight vector w_k - Decision: argmax $P(y=k|x)$ - Relationship to logistic regression (binary is special

case) 23. **Categorical Cross-Entropy** - Loss function for multiclass classification - Formula: $L = -\Sigma_k y_k \log(\hat{y}_k)$ - One-hot encoding: y = [0, 0, 1, 0, ...] for true class - Generalizes binary cross-entropy to K classes - Penalizes deviation from true class distribution - Combined with softmax: differentiable end-to-end - Gradient: $\hat{y}_k - y_k$ (elegant and simple!) 24. **Regularized Logistic Regression** - Prevent overfitting with penalty terms - L2 (Ridge): Loss + $\lambda\Sigma w_i^2$ (smooth coefficient shrinkage) - L1 (Lasso): Loss + $\lambda\Sigma|w_i|$ (feature selection) - Elastic Net: combines L1 and L2 penalties - Hyperparameter λ: controls regularization strength - Especially important with high-dimensional data - Cross-validation to choose optimal λ 25. **Real-World Cases and Implementation** - Applications: spam detection, medical diagnosis, credit scoring - Scikit-learn implementation: `LogisticRegression(penalty='l2', C=1.0)` - Key hyperparameters: regularization (C), solver, max_iter - Evaluation metrics: accuracy, precision, recall, F1-score, ROC-AUC - Important considerations: - Class imbalance: use `class_weight='balanced'` - Feature scaling: standardize for better convergence - Best practices: cross-validation, calibration, threshold tuning #### Key Concepts: - **Maximum Likelihood Estimation:** Statistical method for parameter estimation - **Cross-Entropy Loss:** Probabilistic loss function for classification - **Gradient Descent:** Iterative optimization algorithm - **Softmax Function:** Generalization of sigmoid to multiple classes - **Regularization in Classification:** Preventing overfitting in logistic models #### Learning Outcomes: - Students can derive the logistic regression loss function from MLE - Students can implement gradient descent for logistic regression - Students understand the connection between MLE and cross-entropy - Students can extend binary classification to multiclass problems - Students can apply regularization techniques to prevent overfitting - Students can implement and evaluate logistic regression models in practice --- ## 🛠️ Prerequisites ### Required Background Knowledge: - **Linear Algebra:** Vector operations, matrix multiplication, dot products, norms - **Calculus:** Partial derivatives, gradient computation, chain rule - **Probability Theory:** Conditional probability, likelihood, Bernoulli distribution - **Statistics:** Maximum likelihood estimation, hypothesis testing - **Previous Lectures:** - Lecture 1: Introduction to Machine Learning - Lecture 2: Basic concepts of supervised learning - Lecture 3: Linear regression fundamentals ### Software Requirements: - **Python:** 3.7 or higher - **Essential Libraries:** - NumPy (≥1.19.0) - numerical computing - Pandas (≥1.1.0) - data manipulation - Matplotlib (≥3.3.0) - visualization - Seaborn (≥0.11.0) - statistical visualization - Scikit-learn (≥0.24.0) - machine learning algorithms - Jupyter Notebook or JupyterLab - interactive development ### Optional but Recommended: - **Interactive Tools:** - Desmos graphing calculator (for polynomial regression visualization) - TensorBoard (for training visualization) - **Development Environment:** - VS Code with Python extension - Google Colab (for cloud-based execution) - Anaconda distribution (for package management) ### Installation: ```bash # Using pip pip install numpy pandas matplotlib seaborn scikit-learn jupyter # Using conda conda install numpy pandas matplotlib seaborn scikit-learn jupyter-lab ``` --- ## 💻 Hands-on Components ### 1. Polynomial Regression Interactive Demo **Objective:** Understand how polynomial degree affects model fit and overfitting **Activity:** - Use the Desmos simulator: https://www.desmos.com/calculator/wdb45brrj8?lang=ko - Experiment with degrees 1, 2, 3, 5, 10 - Observe bias-variance trade-off - Identify optimal degree using validation set **Learning Points:** - Visual understanding of underfitting vs. overfitting - Impact of model complexity on training and test error - Importance of cross-validation for model selection ### 2. Regularization Comparison Project **Objective:** Compare Ridge, Lasso, and Elastic Net on real dataset **Dataset:** Boston Housing or California Housing (from scikit-learn) **Tasks:** 1. Load and preprocess data (standardization) 2. Implement Ridge regression with varying λ 3. Implement Lasso regression with varying λ 4. Implement Elastic Net with grid search over α and λ 5. Plot regularization paths for all methods 6. Compare feature selection behavior 7. Evaluate performance using cross-validation **Expected Outputs:** - Regularization path plots (coefficient values vs. λ) - Cross-validation error curves - Selected features for Lasso and Elastic Net - Performance comparison table (R^2, RMSE, MAE) **Code Template:** ```python from sklearn.linear_model import Ridge, Lasso, ElasticNet from sklearn.model_selection import GridSearchCV # Ridge ridge = Ridge() param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]} ridge_cv = GridSearchCV(ridge, param_grid, cv=5) ridge_cv.fit(X_train, y_train) ``` ### 3. Logistic Regression from Scratch **Objective:** Implement logistic regression without using scikit-learn **Components to Implement:** 1. Sigmoid function: `σ(z) = 1 / (1 + exp(-z))` 2. Binary cross-entropy loss function 3. Gradient computation 4. Gradient descent optimization 5. Prediction function 6. Evaluation metrics (accuracy, precision, recall) **Dataset:** Iris dataset (binary classification: Setosa vs. not Setosa) **Validation:** - Compare results with scikit-learn's

LogisticRegression - Convergence plots (loss vs. iteration) - Decision boundary visualization (2D features) **Code Structure:** ```python class LogisticRegressionScratch: def __init__(self, learning_rate=0.01, iterations=1000): self.lr = learning_rate self.iterations = iterations def sigmoid(self, z): # Implement sigmoid function pass def loss(self, y, y_pred): # Implement binary cross-entropy pass def fit(self, X, y): # Implement gradient descent pass def predict(self, X): # Implement prediction pass ``` ### 4. Multiclass Classification Project **Objective:** Implement both One-vs-Rest and Softmax approaches **Dataset:** MNIST digits (subset of 3 classes) or Iris dataset (3 classes) **Tasks:** 1. **One-vs-Rest:** - Train 3 binary classifiers - Combine predictions - Evaluate confusion matrix 2. **Softmax Regression:** - Implement softmax function - Train single multiclass model - Compare with One-vs-Rest 3. **Comparison Analysis:** - Training time comparison - Prediction accuracy comparison - Probability calibration assessment - Visualize decision boundaries (if 2D) **Expected Deliverables:** - Confusion matrices for both approaches - ROC curves (one-vs-rest for each class) - Probability calibration plots - Performance metrics table - Written analysis (1-2 pages) ### 5. Real-World Application: Spam Detection **Objective:** Build production-ready spam classifier **Dataset:** SMS Spam Collection or Enron Email dataset **Pipeline:** 1. **Data Preprocessing:** - Text cleaning (remove punctuation, lowercase) - Tokenization - Stop word removal - TF-IDF vectorization (max_features=3000) 2. **Model Development:** - Train logistic regression with L2 regularization - Hyperparameter tuning (C parameter) - Cross-validation (5-fold) - Handle class imbalance (class_weight='balanced') 3. **Evaluation:** - Accuracy, Precision, Recall, F1-score - ROC-AUC curve - Confusion matrix - Feature importance analysis (top 20 spam/ham words) 4. **Deployment Considerations:** - Model serialization (pickle or joblib) - Inference time measurement - API endpoint design (optional: Flask) **Performance Targets:** - Accuracy: > 95% - F1-score: > 0.93 - False positive rate: < 5% - Inference time: < 50ms per email **Code Example:** ```python from sklearn.feature_extraction.text import TfidfVectorizer from sklearn.linear_model import LogisticRegression from sklearn.metrics import classification_report # Preprocessing vectorizer = TfidfVectorizer(max_features=3000, stop_words='english') X_train_tfidf = vectorizer.fit_transform(X_train) # Model training clf = LogisticRegression(penalty='l2', C=1.0, class_weight='balanced', max_iter=1000, random_state=42) clf.fit(X_train_tfidf, y_train) # Evaluation y_pred = clf.predict(X_test_tfidf) print(classification_report(y_test, y_pred)) ``` --- ## 📖 Additional Resources ### Textbooks: 1. **"An Introduction to Statistical Learning" by James, Witten, Hastie, Tibshirani** - Chapter 4: Classification (Logistic Regression sections) - Chapter 6: Linear Model Selection and Regularization - Freely available: https://www.statlearning.com/ 2. **"The Elements of Statistical Learning" by Hastie, Tibshirani, Friedman** - Chapter 3.4: Shrinkage Methods (Ridge and Lasso) - Chapter 4.4: Logistic Regression - Advanced mathematical treatment - Freely available: https://web.stanford.edu/~hastie/ElemStatLearn/ 3. **"Pattern Recognition and Machine Learning" by Christopher Bishop** - Chapter 3.1: Linear Models for Classification - Chapter 4.3: Probabilistic Discriminative Models - Comprehensive Bayesian perspective ### Online Courses: 1. **Coursera: Machine Learning by Andrew Ng** - Week 3: Logistic Regression and Regularization - Video lectures with excellent intuitions - Programming assignments in Octave/MATLAB 2. **Fast.ai: Practical Deep Learning** - Lesson 4: Natural Language Processing - Shows logistic regression as baseline for text classification ### Research Papers: 1. **"Regularization Paths for Generalized Linear Models via Coordinate Descent"** - Friedman, Hastie, Tibshirani (2010) - Journal of Statistical Software - Efficient algorithms for Lasso and Elastic Net 2. **"LIBLINEAR: A Library for Large Linear Classification"** - Fan et al. (2008) - Scalable implementation details - Journal of Machine Learning Research ### Interactive Tools: 1. **Desmos Polynomial Regression Simulator** - https://www.desmos.com/calculator/wdb45brrj8?lang=ko - Visualize polynomial fits with different degrees 2. **Scikit-learn Documentation** - https://scikit-learn.org/stable/modules/linear_model.html - Comprehensive API reference and examples 3. **ML Playground** - https://ml-playground.com/ - Interactive visualization of classification algorithms ### Video Lectures: 1. **StatQuest with Josh Starmer** - "Logistic Regression Details" series on YouTube - Excellent step-by-step explanations with visual aids 2. **3Blue1Brown: Neural Networks** - Chapter 3: Gradient Descent - Beautiful animations of optimization ### Datasets for Practice: 1. **UCI Machine Learning Repository** - Iris, Wine, Breast Cancer datasets - https://archive.ics.uci.edu/ml/ 2. **Kaggle Datasets** - Titanic: Classification challenge - Credit Card Fraud Detection - https://www.kaggle.com/datasets 3. **Scikit-learn Built-in Datasets** - `load_iris()`, `load_breast_cancer()`, `load_wine()` - Convenient for quick experimentation --- ## 🔗 How to Use These Materials ### For Self-Study: 1. **Week 1: Advanced Linear

Regression (Part 1)** - **Days 1-2:** Review slides 3-6, implement polynomial regression - **Days 3-4:** Study regularization (slides 7-9), complete Regularization Comparison Project - **Day 5:** Work on Feature Selection and Importance (slide 10) - **Weekend:** Review and consolidate understanding 2. **Week 2: Classification Fundamentals (Part 2)** - **Days 1-2:** Study slides 12-16, understand perceptron and sigmoid - **Days 3-4:** Implement Logistic Regression from Scratch project - **Day 5:** Read supplementary materials on MLE and log-odds - **Weekend:** Practice with additional classification problems 3. **Week 3: Advanced Topics and Applications (Part 3)** - **Days 1-2:** Study slides 21-25, understand MLE and gradient descent - **Days 3-4:** Complete Multiclass Classification Project - **Day 5:** Start Real-World Spam Detection project - **Weekend:** Finish Spam Detection and write analysis report ### For Classroom Instruction: **Lecture 1 (90 minutes): Advanced Linear Regression** - **0:00-0:15:** Review previous lecture (slide 4), class discussion - **0:15-0:30:** Polynomial regression (slides 5-6), demo with Desmos - **0:30-0:55:** Regularization methods (slides 7-9), mathematical derivations - **0:55-1:15:** Live coding: Ridge vs. Lasso comparison - **1:15-1:25:** Feature selection (slide 10), case studies - **1:25-1:30:** Q&A and preview of classification - **Homework:** Regularization Comparison Project (due in 1 week) **Lecture 2 (90 minutes): Transition to Classification** - **0:00-0:10:** Review regression limitations (slide 11) - **0:10-0:30:** Classification basics (slides 12-14), group activity - **0:30-0:45:** Perceptron algorithm (slide 15), implementation demo - **0:45-1:05:** Why linear regression fails (slides 16-18), mathematical proof - **1:05-1:25:** Sigmoid function (slides 19-20), properties derivation - **1:25-1:30:** Q&A and MLE preview - **Homework:** Logistic Regression from Scratch (due in 1 week) **Lecture 3 (90 minutes): Logistic Regression Complete** - **0:00-0:15:** MLE derivation (slides 22-23), board work - **0:15-0:30:** Binary cross-entropy (slide 24), loss landscape visualization - **0:30-0:50:** Gradient descent (slide 25), live implementation - **0:50-1:05:** Multiclass extensions (slides 26-28), comparison - **1:05-1:20:** Regularization and real-world cases (slides 29-30) - **1:20-1:30:** Final Q&A, course project introduction - **Final Project:** Real-World Spam Detection (due in 2 weeks) ### Recommended Study Sequence: 1. **Pre-lecture Preparation (30 minutes):** - Read corresponding textbook chapter - Review prerequisite concepts (linear algebra, calculus) - Prepare questions on unclear topics 2. **During Lecture (90 minutes):** - Active note-taking with focus on derivations - Ask questions immediately when confused - Participate in live coding demonstrations 3. **Post-lecture Review (60 minutes):** - Review slides and annotate with additional notes - Work through example problems - Start homework assignment early 4. **Hands-on Practice (2-4 hours per week):** - Complete programming assignments - Experiment with different parameters - Analyze results and write interpretations 5. **Weekly Review (30 minutes):** - Summarize key concepts in own words - Create concept map connecting topics - Identify areas needing clarification ### Interactive Elements: - **Desmos Polynomial Regression:** Use during Part 1 to visualize overfitting in real-time - **Live Coding Sessions:** Implement algorithms step-by-step during lecture - **Think-Pair-Share:** Discuss classification problems and model choices in small groups - **Whiteboard Derivations:** Work through MLE and gradient descent on board collaboratively --- ## 📊 Assessment Suggestions ### Formative Assessment (During Learning): 1. **Quick Polls (5 questions per lecture):** - Example: "Which regularization method produces sparse models?" (Lasso) - Example: "What is the output range of sigmoid function?" ((0,1)) - Use tools like Mentimeter or Kahoot for engagement 2. **Concept Check Questions:** - After slide 9: "Explain the difference between L1 and L2 penalties in your own words" - After slide 15: "Why does perceptron fail on non-linearly separable data?" - After slide 24: "Derive the gradient of binary cross-entropy loss" 3. **Peer Instruction:** - Present common misconceptions and have students vote - Example: "True or False: Lasso always performs better than Ridge" (False) - Discuss in pairs, revote, then instructor explains ### Summative Assessment (Final Evaluation): **Programming Assignment (40% of grade):** *Assignment: Comprehensive Classification Pipeline* **Part A: Regularization (15 points)** - Implement Ridge, Lasso, Elastic Net on provided dataset - Perform hyperparameter tuning with cross-validation - Plot regularization paths and interpret results - Write 1-page analysis of feature selection behavior **Part B: Logistic Regression from Scratch (15 points)** - Implement complete logistic regression (sigmoid, loss, gradient, training) - Achieve convergence (loss decreases monotonically) - Match scikit-learn performance (within 2% accuracy) - Visualize decision boundary for 2D case **Part C: Real-World Application (10 points)** - Build spam detection system with full pipeline - Achieve F1-score > 0.90 on test set - Provide confusion matrix and ROC-AUC curve - Deploy as simple Flask API (optional for extra credit) **Grading Rubric:** - Code quality and documentation

(20%) - Correctness of implementation (40%) - Experimental results and visualizations (25%) - Written analysis and interpretation (15%) **Written Exam (30% of grade):** *Section 1: Conceptual Understanding (15 points)* 1. Explain bias-variance trade-off in context of polynomial regression (5 pts) 2. Derive sigmoid function from log-odds transformation (5 pts) 3. Compare One-vs-Rest and Softmax for multiclass classification (5 pts) *Section 2: Mathematical Derivations (10 points)* 1. Derive gradient of binary cross-entropy loss (5 pts) 2. Show that Lasso penalty leads to sparse solutions (5 pts) *Section 3: Applied Problems (5 points)* 1. Given confusion matrix, calculate precision, recall, F1-score (3 pts) 2. Analyze regularization path plot and explain feature importance (2 pts) **Project Presentation (20% of grade):** *Format: 10-minute presentation + 5-minute Q&A* **Requirements:** - Problem definition and dataset description (2 points) - Methodology and model selection rationale (5 points) - Results presentation with visualizations (5 points) - Critical analysis and limitations discussion (5 points) - Code demonstration and reproducibility (3 points) **Evaluation Criteria:** - Technical depth and accuracy - Clarity of explanation and visualization quality - Ability to answer questions and defend choices - Creativity in problem-solving approaches **Participation and Quizzes (10% of grade):** - Weekly quizzes on Canvas/Moodle (5 short questions, 5 points each) - In-class participation and discussions (subjective, 50 points) ### Suggested Topics for Final Project: 1. **Medical Diagnosis:** Predict disease presence from clinical features 2. **Customer Churn:** Predict customer retention in subscription service 3. **Sentiment Analysis:** Classify movie reviews as positive/negative 4. **Credit Risk:** Predict loan default probability 5. **Image Classification:** Classify handwritten digits (MNIST subset) 6. **Fraud Detection:** Identify fraudulent transactions in credit card data ### Grading Scale: - A: 90-100% (Exceptional understanding and implementation) - B: 80-89% (Strong understanding with minor errors) - C: 70-79% (Adequate understanding, some conceptual gaps) - D: 60-69% (Significant conceptual gaps) - F: <60% (Insufficient understanding) --- ## 📝 Notes for Implementation ### Technical Requirements: **Computational Resources:** - **Minimum:** 4GB RAM, 2-core CPU, 10GB storage - **Recommended:** 8GB RAM, 4-core CPU, 20GB storage - **For large datasets:** GPU acceleration (CUDA-compatible) recommended but not required - Cloud alternatives: Google Colab (free GPU), Kaggle Kernels **Software Versions (Tested):** - Python 3.8.10 - NumPy 1.21.2 - Pandas 1.3.3 - Matplotlib 3.4.3 - Seaborn 0.11.2 - Scikit-learn 1.0.1 - Jupyter Lab 3.2.1 **Installation Verification:** ```python import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns import sklearn print(f"NumPy: {np.__version__}") print(f"Pandas: {pd.__version__}") print(f"Matplotlib: {matplotlib.__version__}") print(f"Seaborn: {sns.__version__}") print(f"Scikit-learn: {sklearn.__version__}") ``` ### Common Issues and Solutions: **Issue 1: Sigmoid Overflow** - **Problem:** `exp(-z)` causes overflow for large negative z - **Solution:** Use stable sigmoid implementation: ```python def sigmoid_stable(z): return np.where(z >= 0, 1 / (1 + np.exp(-z)), np.exp(z) / (1 + np.exp(z))) ``` **Issue 2: Gradient Descent Not Converging** - **Causes:** Learning rate too large, features not standardized, poor initialization - **Solutions:** - Scale features: `StandardScaler()` or `MinMaxScaler()` - Reduce learning rate: try η = 0.01, 0.001, 0.0001 - Use adaptive methods: Adam, RMSprop - Check gradient computation (compare with numerical gradient) **Issue 3: Class Imbalance** - **Problem:** 90% class 0, 10% class 1 → model predicts all class 0 - **Solutions:** - Use `class_weight='balanced'` in LogisticRegression - Oversample minority class (SMOTE) - Undersample majority class - Adjust decision threshold (not just 0.5) - Use appropriate metrics (F1, precision-recall, not just accuracy) **Issue 4: Overfitting in High-Dimensional Data** - **Symptoms:** Perfect training accuracy, poor test accuracy - **Solutions:** - Increase regularization (decrease C parameter) - Use L1 penalty for feature selection - Reduce feature dimensionality (PCA, feature selection) - Collect more training data **Issue 5: Memory Issues with Large Datasets** - **Problem:** Out of memory error with large datasets (>1M samples) - **Solutions:** - Use mini-batch gradient descent (batch_size=256 or 512) - Employ SGDClassifier instead of LogisticRegression - Use sparse matrices for text data (`scipy.sparse`) - Process data in chunks with `pandas.read_csv(chunksize=...)` ### Performance Optimization Tips: 1. **Vectorization:** - Always use NumPy operations instead of Python loops - Bad: `for i in range(n): y[i] = sigmoid(X[i].dot(w))` - Good: `y = sigmoid(X.dot(w))` 2. **Efficient Regularization:** - Don't regularize bias term (only weights) - Use warm-start in scikit-learn for hyperparameter search 3. **Early Stopping:** - Monitor validation loss and stop if no improvement for k epochs - Saves computation time and prevents overfitting 4. **Parallel Processing:** - Use `n_jobs=-1` in scikit-learn to use all CPU cores - Especially beneficial for cross-validation 5. **Data Loading:** - Use `pandas.read_csv(dtype=...)` to specify data types - Reduces memory usage by 50-70% for

large datasets ### Debugging Checklist: - [ ] Features are standardized (mean=0, std=1) - [ ] Labels are binary (0/1) for binary classification - [ ] No NaN or infinite values in data - [ ] Sigmoid function is numerically stable - [ ] Loss is decreasing (plot loss vs. iteration) - [ ] Gradient is computed correctly (numerical check) - [ ] Learning rate is appropriate (not too large/small) - [ ] Regularization strength is reasonable (C between 0.01 and 100) - [ ] Test set is never used for training or hyperparameter tuning - [ ] Random seed is set for reproducibility ### Best Practices: 1. **Reproducibility:** - Set random seeds: `np.random.seed(42)`, `random_state=42` - Document package versions in `requirements.txt` - Use version control (Git) for code 2. **Code Organization:** - Separate data preprocessing, model training, evaluation - Use functions and classes, not just scripts - Add docstrings and type hints - Follow PEP 8 style guide 3. **Experiment Tracking:** - Log hyperparameters and results (use Weights & Biases or MLflow) - Save trained models with `joblib` or `pickle` - Version datasets and document preprocessing steps 4. **Visualization Standards:** - Always label axes and include titles - Use colorblind-friendly palettes (`sns.color_palette("colorblind")`) - Export plots at high resolution (300 DPI) for reports - Include legends when plotting multiple series 5. **Documentation:** - README with setup instructions and usage examples - Jupyter notebooks with markdown explanations - Inline comments for complex code sections - Results summary with tables and visualizations --- ## 🙏 Credits **Instructor:** Ho-min Park **Affiliation:** Ghent University Global Campus (GUGC) **Email:** homin.park@ghent.ac.kr | powersimmani@gmail.com **Acknowledgments:** - Course materials build upon fundamental concepts from statistical learning theory - Interactive Desmos simulator created for enhanced student engagement - Slide design follows modern educational presentation best practices - Code examples tested on scikit-learn 1.0+ for compatibility **Citation:** ``` Park, H. (2025). Lecture 4: From Linear to Logistic Regression. Machine Learning Course Materials. Ghent University Global Campus. ``` **Version:** 1.0 **Last Updated:** 2025 **Course:** Introduction to Machine Learning / Data Science Fundamentals --- **Feedback and Questions:** For questions about lecture content, implementation issues, or suggestions for improvement, please contact the instructor via email. Office hours and additional support sessions can be arranged upon request. **Course Repository:** Complete code examples, datasets, and additional resources are available in the accompanying course repository. Students are encouraged to contribute improvements via pull requests. --- *This README is designed to be comprehensive yet accessible, providing clear pathways for both self-study and classroom instruction. The structure supports diverse learning styles through multiple modalities: visual (slides), theoretical (derivations), and practical (coding projects).*

Eigenvalues and Eigenvectors

Eigenvector v

$$Av = \lambda v$$

Direction unchanged by matrix A

Eigenvalue λ

Scaling factor for the eigenvector

Characteristic Equation

$\det(A - \lambda I) = 0$

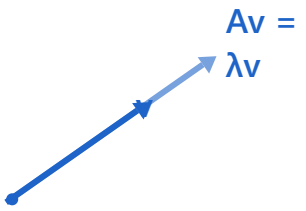
Number of Eigenvalues

n×n matrix has n eigenvalues
(counting multiplicity)

Spectral Theorem

Symmetric matrices have
orthogonal eigenvectors

Geometric Interpretation



Matrix A **stretches** v by factor λ
but keeps the **same direction**

Eigendecomposition

$$A = Q\Lambda Q^T$$

For symmetric matrix A
Q: orthogonal eigenvectors
Λ: diagonal matrix of eigenvalues

ML Applications

- Understanding data variance and covariance structure
- PCA (Principal Component Analysis)
- Regression diagnostics use eigenanalysis

Example: Finding Eigenvalues and Eigenvectors

Given Matrix A

$$A = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix}$$

Step 1: Find eigenvalues using $\det(A - \lambda I) = 0$

$$\det\begin{bmatrix} 4-\lambda & 1 \\ 2 & 3-\lambda \end{bmatrix} = 0$$

$$(4-\lambda)(3-\lambda) - 2 = 0$$
$$\lambda^2 - 7\lambda + 10 = 0$$
$$(\lambda-5)(\lambda-2) = 0$$

$\lambda_1 = 5, \lambda_2 = 2$

Find Eigenvectors

For $\lambda_1 = 5$: Solve $(A - 5I)v = 0$

$$\begin{bmatrix} -1 & 1 \\ 2 & -2 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$-v_1 + v_2 = 0 \rightarrow v_2 = v_1$$

$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

For $\lambda_2 = 2$: Solve $(A - 2I)v = 0$

$$\begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$2v_1 + v_2 = 0 \rightarrow v_2 = -2v_1$$

$v_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$

Differentiation and Partial Derivatives

Derivative f'(x)

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Rate of change at a point

Partial Derivative ∂f/∂x

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x+he_i) - f(x)}{h}$$

Derivative w.r.t. one variable, holding others constant

Chain Rule

$$\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$$

Essential for backpropagation

Product Rule

$$(fg)' = f'g + fg'$$

Quotient Rule

$$\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$$

Gradient ∇f(x)

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right]^T$$

Vector of all partial derivatives

Points in direction of steepest ascent

Magnitude = rate of increase

Hessian Matrix H(f)

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

Matrix of second derivatives

Captures curvature of function

Optimization Condition

At minimum/maximum:

$$\nabla f(x^*) = 0$$

ML Applications

- Gradient descent optimization
- Finding regression coefficients
- Minimizing loss functions

- $\partial L/\partial \beta = 0$ solves for optimal β

Example: Computing Derivatives and Gradient

Example 1: Chain Rule

Given: $f(x) = (3x^2 + 2x)^5$

Find: $f'(x)$

Solution: Let $u = 3x^2 + 2x$

$$f(x) = u^5$$

$$f'(x) = 5u^4 \cdot u'$$

$$u' = 6x + 2$$

$$f'(x) = 5(3x^2 + 2x)^4(6x + 2)$$

Example 2: Partial Derivatives

Given: $f(x,y) = x^2y + 3xy^2 + y^3$

Find: $\partial f/\partial x$ and $\partial f/\partial y$

$\partial f/\partial x$: (treat y as constant)

$$\partial f/\partial x = 2xy + 3y^2$$

$\partial f/\partial y$: (treat x as constant)

$$\partial f/\partial y = x^2 + 6xy + 3y^2$$

$$\nabla f = [2xy + 3y^2, x^2 + 6xy + 3y^2]^T$$

Example 3: Gradient Descent Step

Loss function: $L(w) = (w - 3)^2$

Current: $w = 0$, learning rate $\alpha = 0.1$

Step 1: Compute gradient

$$\nabla L(w) = 2(w - 3)$$

$$\nabla L(0) = 2(0 - 3) = -6$$

Step 2: Update parameter

$$w\_new = w - \alpha \nabla L(w)$$

$$w\_new = 0 - 0.1(-6) = 0.6$$

New $w = 0.6$ (closer to minimum at $w = 3$)

Example 4: Hessian Matrix

Given: $f(x,y) = x^2 + 2xy + 3y^2$

Find: Hessian $H(f)$

First derivatives:

$$\partial f/\partial x = 2x + 2y$$

$$\partial f/\partial y = 2x + 6y$$

Second derivatives:

$$\partial^2 f/\partial x^2 = 2, \quad \partial^2 f/\partial x \partial y = 2$$

$$\partial^2 f/\partial y \partial x = 2, \quad \partial^2 f/\partial y^2 = 6$$

H = [2 2]
[2 6]

Part 2/3:

Probability and Statistics Fundamentals

10. Probability Spaces and Random Variables
11. Probability Distributions - Discrete and Continuous
12. Expectation, Variance, and Covariance
13. Conditional Probability and Bayes' Theorem
14. Central Limit Theorem and Law of Large Numbers
15. Parameter Estimation - MLE and MAP
16. Hypothesis Testing and Confidence Intervals
17. Correlation vs Causation

Probability Spaces and Random Variables

Sample Space Ω

Set of all possible outcomes

Event A

Subset of sample space

$$A \subseteq \Omega$$

Probability Measure P

$$P(A) \in [0,1]$$

$$P(\Omega) = 1$$

Random Variable X

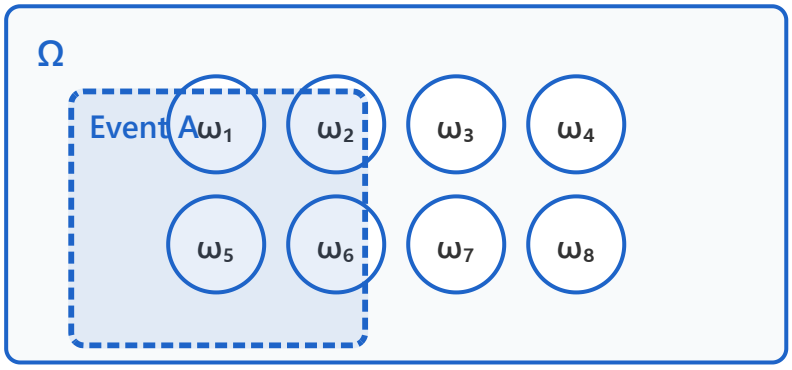
Function mapping outcomes to real numbers

Types of Random Variables

Discrete RV

Countable values (coin flips, dice)

Sample Space & Event Example



Random Variable X: Ω → ℝ

Outcomes		Values
Heads	→	1
Tails		0

CDF: F(x) = P(X ≤ x)

Cumulative distribution function

Continuous RV

Uncountable values (heights, temps)

ML Application

Foundation for modeling uncertainty in regression

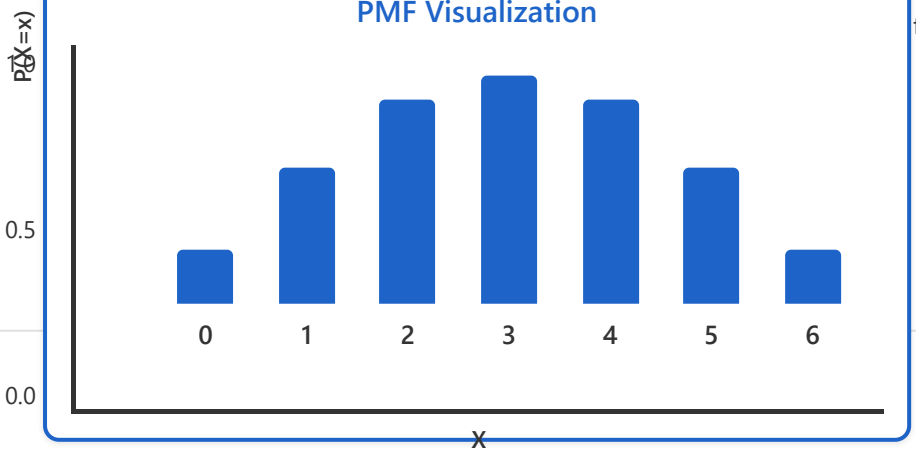
Probability Distributions - Discrete and Continuous

Discrete Distributions
Probability Mass Function (PMF)

P(X = x)
Probability of specific values

Examples
- **Bernoulli:** Coin flip
- **Binomial:** n trials
- **Poisson:** Rare events

PMF Visualization



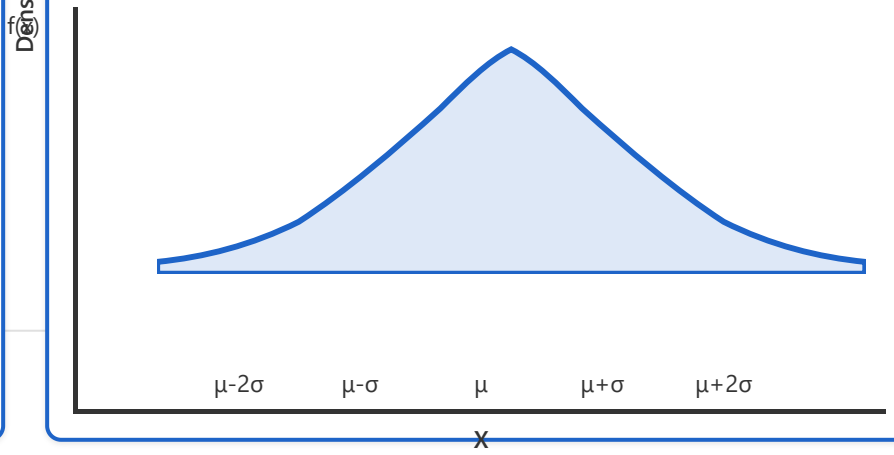
Continuous Distributions
Probability Density Function (PDF)

f(x)
$P(a \le X \le b) = \int_a^b f(x)dx$

Normal Distribution
$$X \sim N(\mu, \sigma^2)$$

Bell-shaped curve

PDF Visualization (Normal)



Standard Normal Distribution

ML Impact

Z ~ N(0, 1) - Used for standardization

Errors in regression often assumed normal

Enables statistical inference

Expectation, Variance, and Covariance

Expectation E[X]

Average value

$$E[X] = \Sigma\ xP(X=x) \text{ or } \int xf(x)dx$$

Properties

$$E[aX + b] = aE[X] + b$$

$$E[X + Y] = E[X] + E[Y]$$

Variance Var(X)

Spread around mean

$$Var(X) = E[(X - \mu)^2]$$

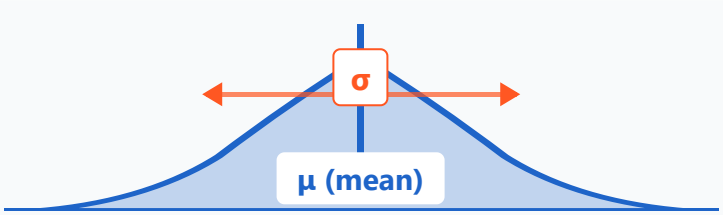
Standard Deviation σ

$$\sigma = \sqrt{Var(X)}$$

Covariance

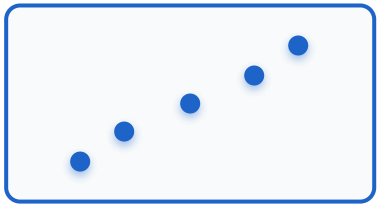
$$Cov(X,Y) = E[(X - \mu_X)(Y - \mu_Y)]$$

Variance: Spread Around Mean

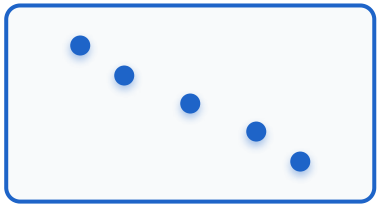


Covariance Sign Interpretation

Positive Cov



Negative Cov



Covariance Matrix

$$\Sigma = [Cov(X_i, X_j)]$$

For multiple variables

Regression Application

Positive: X and Y increase together

Understanding variable relationships

Conditional Probability and Bayes' Theorem

Conditional Probability

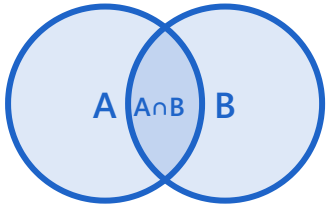
$$P(A|B) = P(A \cap B) / P(B)$$

Probability of A given B has occurred

Multiplication Rule

$$P(A \cap B) = P(A|B)P(B)$$

Visual Representation



Independence: X ⊥ Y

$$P(X|Y) = P(X)$$

Bayes' Theorem

$$P(A|B) = P(B|A)P(A) / P(B)$$

Prior	→	Likelihood	→	Posterior
$P(A)$		$P(B\|A)$		$P(A\|B)$

Applications

- Updating beliefs with new data
- Medical diagnosis and testing
- Spam filtering and classification

ML Foundation

Bayesian regression and inference

Central Limit Theorem and Law of Large Numbers

Law of Large Numbers

Sample mean converges to population mean

$$\bar{X}_n \to \mu \text{ as } n \to \infty$$

Convergence in probability

Central Limit Theorem

Sum of random variables approaches normal distribution

$$\sqrt{n}(\bar{X}_n - \mu) / \sigma \to N(0,1)$$

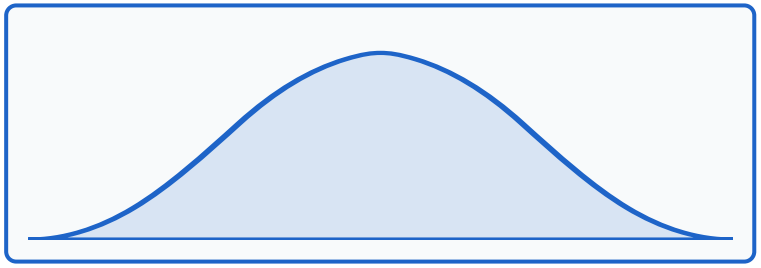
as $n \to \infty$

CLT: Any Distribution → Normal Distribution

Original Distribution
(any shape)

Take means
$n \to \infty$

Distribution of Means
(normal)



Key Insight

Averages are approximately normal with large n

Regression Use

Justifies normal assumption in residuals

Enables

Confidence intervals & hypothesis tests

Real-World Applications

Quality Control

A/B Testing

Opinion Polls

Manufacturing defect rates follow normal distribution with large samples.

$$\bar{X} \sim N(\mu, \sigma^2/n)$$

Use: *Setting acceptable quality limits in production lines*

Website conversion rates comparison uses CLT for statistical significance.

$$\hat{p} \sim N(p, p(1-p)/n)$$

Use: *Deciding which website design performs better*

Survey results become normally distributed with sufficient sample size.

$$\text{Margin of Error} = 1.96 \times \sigma/\sqrt{n}$$

Use: *Predicting election outcomes with confidence intervals*

Financial Risk

Portfolio returns approximate normal distribution over time.

$$R_{portfolio} \sim N(\mu_p, \sigma_p^2)$$

Use: *Value-at-Risk (VaR) calculations for investments*

Medical Trials

Average treatment effects tested using normal approximation.

$$t = (\bar{X} - \mu_0)/(s/\sqrt{n})$$

Use: *Determining if new drugs are effective*

Machine Learning

Bootstrap confidence intervals rely on CLT for model evaluation.

$$Accuracy \sim N(\mu_{acc}, \sigma^2_{acc}/n)$$

Use: *Estimating model performance reliability*

Parameter Estimation - MLE and MAP

Parameter Estimation: Inferring θ from observed data

Maximum Likelihood Estimation (MLE)

Objective

$$\text{argmax } L(\theta|\text{data})$$

Likelihood Function

$$L(\theta) = P(\text{data}|\theta) = \prod_i P(x_i|\theta)$$

Log-Likelihood

$$\ell(\theta) = \log L(\theta)$$

Easier to optimize

MLE for Normal Distribution

$$\hat{\mu} = \bar{X}$$

$$\sigma^2 = (1/n)\Sigma(x_i - \bar{X})^2$$

Maximum A Posteriori Estimation (MAP)

Objective

$$\text{argmax } P(\theta|\text{data})$$

Using Bayes' Theorem

$$P(\theta|\text{data}) = P(\text{data}|\theta)P(\theta) / P(\text{data})$$

MAP = MLE + Prior

Incorporates prior knowledge

Key Difference

Quick Comparison

Aspect	MLE	MAP
Prior	No prior	Uses prior P(θ)
Focus	Data only	Data + knowledge

Regression Use

MLE/MAP estimate regression coefficients

Hypothesis Testing and Confidence Intervals

Hypothesis Testing

H_0: Null Hypothesis

Default assumption

$$e.g., \beta = 0$$

H_1: Alternative Hypothesis

Claim to test

$$e.g., \beta \neq 0$$

Test Statistic

Measure computed from data (t-stat, z-stat)

p-value

P(observe data or more extreme | H_0 true)

Decision Process

Set significance level α
(typically 0.05)

↓

Compute test statistic

↓

Confidence Intervals

95% Confidence Interval

$$[\hat{\theta} - 1.96SE, \hat{\theta} + 1.96SE]$$

Calculate p-value

↓

p-value < α ?

Yes: Reject H_0

No: Fail to reject H_0

Interpretation

Range likely containing true parameter

95% of such intervals contain θ

Connection to Testing

If H_0: $\theta = \theta_0$ is not in the CI, then reject H_0 at $\alpha = 0.05$

Real-World Examples: Hypothesis Testing

Case 1: Drug Efficacy

A pharmaceutical company tests whether a new drug reduces blood pressure more than placebo.

H_0: μ_drug = μ_placebo
H_1: μ_drug < μ_placebo

Data:
n = 100 patients
Mean difference = -8.5 mmHg
SE = 2.1 mmHg
t-stat = -4.05

p-value: 0.0001
α level: 0.05

Reject H_0
Drug is effective!

Case 2: A/B Testing

An e-commerce site tests whether a new checkout design increases conversion rate.

H_0: p_new = p_old
H_1: p_new ≠ p_old

Data:
Control: 450/10000 = 4.5%
Treatment: 520/10000 = 5.2%
z-stat = 2.28

p-value: 0.023
α level: 0.05

Reject H_0
New design works!

Case 3: Salary Regression

Testing whether years of education significantly predict salary (β_1 coefficient).

H_0: $\beta_1 = 0$
H_1: $\beta_1 \neq 0$

Data:
n = 500 employees
$\hat{\beta}_1$ = 5,200 ($/year)
SE($\hat{\beta}_1$) = 800
t-stat = 6.5

p-value: < 0.001
95% CI: [3,632, 6,768]

Reject H_0
Education matters!

Correlation vs Causation

Correlation

Definition

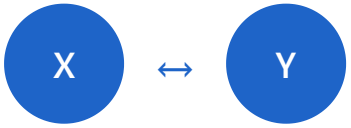
Statistical association between variables

Pearson Correlation

$$\rho = \mathrm{Cov}(X,Y) \;/\; (\sigma_x \sigma_Y)$$

$\rho = -1$	$\rho = 0$	$\rho = 1$
Perfect negative	No linear	Perfect positive

Association only



Causation

Definition

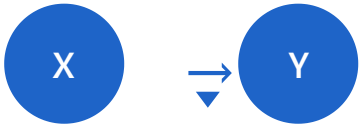
X directly influences Y

Causal Relationship

Requires:

- Temporal precedence
- Mechanism
- Control of confounders

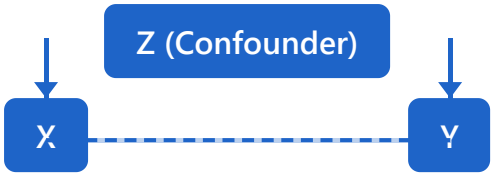
Direct influence



⚠ Key Warning

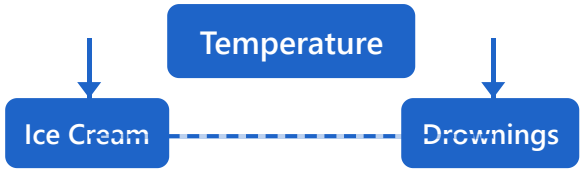
Correlation does NOT imply causation

Confounding Variable



Z affects both X and Y,
creating spurious correlation

Classic Example



Correlated, but ice cream
doesn't cause drownings

Regression Limitation

Regression shows association, not necessarily causation

Part 3/3:

Linear Regression Model

18. Linear Regression Problem Definition
19. Simple Linear Regression Model
20. Deriving the Least Squares Method
21. Normal Equation Solution
22. Geometric Interpretation
23. Multiple Linear Regression Extension
24. Model Assumptions and Diagnostics
25. Python Implementation and Practice

Linear Regression Problem Definition

Goal

Model relationship between input X and output Y

Core Assumption

Linear relationship

$$Y = f(X) + \varepsilon$$

Training Data

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

n labeled examples

Objective

Find function f that best approximates the true relationship

Learning Paradigm

Supervised Learning:
Learning from labeled examples

Regression Workflow

1. Collect Data

Training examples (X, Y)

↓

2. Learn Function f

Find best fit to data

↓

3. Predict

Given new x → estimate $\hat{y} = f(x)$

Applications

- Price prediction
- Trend forecasting
- Causal inference

Bridge from theory to practical machine learning

Matrix Representation

Linear Model

For multiple features:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \varepsilon$$

In matrix form:

$$Y = X\beta + \varepsilon$$

where:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{1\,1} & x_{1\,2} & \cdots & x_{1\,p} \\ 1 & x_{2\,1} & x_{2\,2} & \cdots & x_{2\,p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \quad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Y: n×1, X: n×(p+1), β: (p+1)×1, ε: n×1

Least Squares Solution

Objective: Minimize sum of squared errors

$$\min ||Y - X\beta||^2$$

Normal Equation:

$$X^T X\beta = X^T Y$$

Solution:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Prediction: $\hat{Y} = X\hat{\beta}$

This requires $X^T X$ to be invertible
(X has full column rank)

Simple Linear Regression Model

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

β_0: Intercept

Value when X = 0

β_1 : Slope

Change in Y per unit change in X

ε: Error Term

Random error, $\varepsilon \sim N(0, \sigma^2)$

Fitted Line

$$\hat{y} = \hat{\beta_0} + \hat{\beta_1} x$$

Estimated parameters

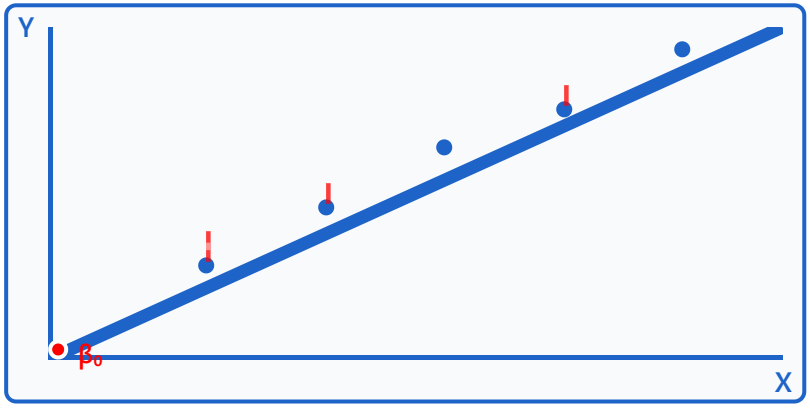
Residual: $e_i = y_i - \hat{y}_i$

Prediction error

Objective

Minimize total prediction error

Visual Representation



Blue line: fitted regression | Dashed red: residuals

Real-World Example

$$\text{Salary} = \beta_0 + \beta_1 \times (\text{Years of Experience}) + \varepsilon$$

📊 2D Interactive Simulator

Explore simple linear regression with an interactive 2D simulator!

Launch 2D Simulator →

🎯 3D Interactive Simulator

Visualize multiple regression in 3D space with interactive controls!

Launch 3D Simulator →

Deriving the Least Squares Method

Loss Function: Sum of Squared Errors

$$L(\beta_0, \beta_1) = \sum_i (y_i - \beta_0 - \beta_1 x_i)^2$$

SSE: Sum of Squared Errors

Why Squares?

- Penalizes large errors more heavily
- Mathematically convenient (differentiable)
- No cancellation of positive/negative errors

Optimization Goal

Find $\hat{\beta}_0, \hat{\beta}_1$ that minimize L

Key Requirement

Unique solution exists when
X has full rank

Statistical Connection

Derivation Steps

1 Define Loss Function

$$L(\beta_0, \beta_1) = \sum (y_i - \beta_0 - \beta_1 x_i)^2$$

↓

2 Take Partial Derivatives

$$\partial L / \partial \beta_0 = 0$$

$$\partial L / \partial \beta_1 = 0$$

↓

3 Solve System of Equations

Normal Equations

↓

✓ Solution

Optimal parameters $\hat{\beta}_0, \hat{\beta}_1$
that minimize prediction error

Least Squares = Maximum Likelihood
under normal errors

Normal Equation Solution

Closed-form solution for linear regression

Matrix Form

$$Y = X\beta + \varepsilon$$

X is the design matrix

↓

Normal Equation

$$X^TX\beta = X^TY$$

↓

General Solution

$$\hat{\beta} = (X^TX)^{-1}X^TY$$

When X^TX is invertible

Simple Regression

$$\hat{\beta_1} = \text{Cov}(X,Y) \,/\, \text{Var}(X)$$

$$\hat{\beta_0} = \bar{Y} - \hat{\beta_1}\,\bar{X}$$

Complexity

Computational cost

$$O(np^2 + p^3)$$

n: samples
p: features

Alternative

For large-scale:
Gradient Descent
(iterative method)

Advantage

Closed-form
Direct computation
No iterations needed

Gradient Descent Process

Parameter update with MSE Loss - Step by Step

Mean Squared Error (MSE) Loss

$$L(\beta) = (1/n)\Sigma(y_i - \hat{y}_i)^2 = (1/n)\Sigma(y_i - \beta_0 - \beta_1 x_i)^2$$

↓

1 Compute Partial Derivatives

$$\partial L/\partial \beta_0 = -(2/n)\Sigma(y_i - \beta_0 - \beta_1 x_i)$$

$$\partial L/\partial \beta_1 = -(2/n)\Sigma x_i(y_i - \beta_0 - \beta_1 x_i)$$

Calculate gradient for each parameter

2 Evaluate at Current Parameters

$$g_0 = \partial L/\partial \beta_0 | \beta^{(t)}, \quad g_1 = \partial L/\partial \beta_1 | \beta^{(t)}$$

Substitute current $\beta_0^{(t)}$ and $\beta_1^{(t)}$ values

3 Update Parameters

$$\beta_0^{(t+1)} = \beta_0^{(t)} - \alpha \cdot g_0$$

$$\beta_1^{(t+1)} = \beta_1^{(t)} - \alpha \cdot g_1$$

α is the learning rate (step size)

4 Check Convergence

$$|L^{(t+1)} - L^{(t)}| < \varepsilon \text{ or } ||\nabla L|| < \varepsilon$$

Repeat steps 1-3 until convergence criterion is met

Learning Rate α

Controls step size

Too large: diverge

Too small: slow

Matrix Form

$$\beta^{(t+1)} = \beta^{(t)} - \alpha \cdot \nabla L$$

$$\nabla L = -(2/n)X^T(Y - X\beta)$$

Convergence

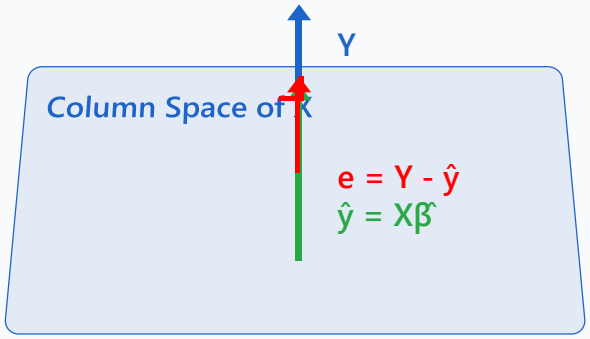
Iterative approach

Works for large-scale

O(np) per iteration

Geometric Interpretation

Projection onto Column Space



Column Space

All possible linear combinations of predictors

Prediction ŷ

Projection of Y onto column space of X

Residual e

Orthogonal to column space (perpendicular)

Projection Matrix

$$P = X(X^TX)^{-1}X^T$$

$$\hat{y} = PY$$

Residual Maker

$$M = I - P$$

$$e = MY$$

Orthogonality

$$\langle e, X_j \rangle = 0$$

for all predictors

$\hat{y}$ is the closest point in column space to Y

Unifies **Linear Algebra** and **Statistics**

Multiple Linear Regression Extension

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon$$

Multiple predictors model

Simple vs Multiple Regression

Aspect	Simple
Predictors	1 variable
Model	$Y = \beta_0 + \beta_1 X$
Aspect	**Multiple**
Predictors	p variables
Model	$Y = X\beta$

Matrix Notation

$$Y = X\beta + \varepsilon$$

$$X \in \mathbb{R}^{n \times (p+1)}$$

Same Solution

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Normal equation applies

Model Fit Metrics

R^2 (R-squared)

$$R^2 = 1 - SSE/SST$$

Proportion of variance explained

Adjusted R^2

Penalizes model complexity

⚠ Multicollinearity

Correlated predictors
cause instability in estimates

Challenges

- Increased complexity
- More parameters to estimate
- Risk of overfitting
- Interpretation becomes harder

Feature Selection

Choosing relevant predictors
to improve model performance

Coefficient Interpretation

β_j is the effect of X_j
holding all other predictors constant

Understanding Multicollinearity in Detail

📊 Definition

High correlation between two or more predictor variables

$$Corr(X_i, X_j) \approx \pm 1$$

Makes it difficult to isolate individual effects

⚠️ Problems

1. Unstable estimates:
$\hat{\beta}$ coefficients become highly sensitive to small data changes

2. Large standard errors:
Var($\hat{\beta}$) increases dramatically

3. Poor interpretability:
Cannot determine which variable is truly important

$(X^TX)^{-1}$ becomes ill-conditioned

🔍 Detection Methods

1. Correlation Matrix:
Check pairwise correlations

$$|corr| > 0.8 \rightarrow \text{Warning}$$

2. VIF (Variance Inflation Factor):

$$VIF = 1/(1 - R^2_j)$$

R^2_j: R^2 when X_j is regressed on other predictors

VIF > 10 → Serious multicollinearity
VIF > 5 → Moderate concern

💡 Solutions

1. Remove variables:
Drop one of the highly correlated predictors

2. Combine variables:
Create composite index (e.g., PCA)

3. Regularization:
Ridge regression (L2) or Lasso (L1)

$$\text{Ridge: } \min ||Y - X\beta||^2 + \lambda||\beta||^2$$

📝 Example Scenario

Predicting house price with:

- Square footage
- Number of rooms

These are highly correlated!
Larger houses → more rooms

Solution: Use only one,
or create "size index"

🎯 Key Insights

Important: Multicollinearity affects *interpretation* and *stability*, but NOT prediction accuracy

The model can still predict well, but we cannot trust individual coefficient values

Overall model fit (R^2) remains good
Individual t-tests become unreliable

Model Assumptions and Diagnostics

Key Assumptions

1. Linearity
True relationship is linear (or approximately)

2. Independence
Observations are independent

3. Homoscedasticity
Constant error variance across X

4. Normality
Errors ~ $N(0, \sigma^2)$

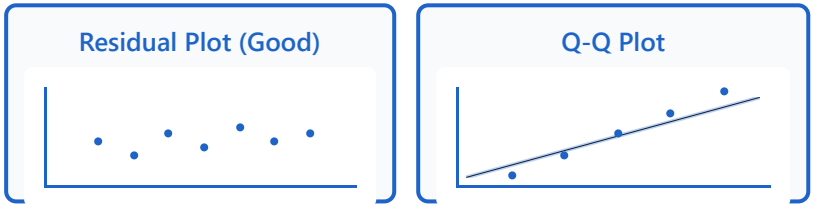
5. No Multicollinearity
Predictors not highly correlated

Diagnostic Tools

- Residual Plots
- Q-Q Plots
- Leverage Plots
- VIF Scores

Visual Diagnostics

Example Diagnostic Plots



Random scatter in residual plot = Good fit
Points on diagonal in Q-Q = Normal errors

Violations & Remedies

- Transformations (log, sqrt)
- Robust regression methods
- Regularization (Ridge, Lasso)

Model Validation

- Train-Test Split
- Cross-Validation
- Bootstrap

Python Implementation and Practice

Key Python Libraries

NumPy
Matrix operations

Scikit-learn
ML convenience functions

Pandas
Data manipulation

Statsmodels
Statistical output

Matplotlib/Seaborn
Visualization

Implementation Examples

```
# NumPy (from scratch)
β = np.linalg.inv(X.T @ X) @ X.T @ y
```

```
# Scikit-learn (easy)
from sklearn.linear_model import LinearRegression
```

Standard Workflow

1. Load Data
↓
2. Explore & Visualize
↓
3. Fit Model
↓
4. Validate (diagnostics)
↓
5. Predict

Best Practices

- Feature scaling
- Handle missing data

```
model = LinearRegression()
model.fit(X, y)
```

```
# Statsmodels (detailed stats)
import statsmodels.api as sm
results = sm.OLS(y, X).fit()
```

Train-test split

Cross-validation

Practice Datasets

Boston Housing

California Housing

Diabetes

Thank you

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com