Foundations of Data Science[1]

Avrim Blum, John Hopcroft, and Ravindran Kannan

Thursday 4th January, 2018

Contents

1 Introduction 9

2 High-Dimensional Space 12

2.1 Introduction 12

2.2 The Law of Large Numbers 12

2.3 The Geometry of High Dimensions 15

2.4 Properties of the Unit Ball 17

2.4.1 Volume of the Unit Ball 17

2.4.2 Volume Near the Equator 19

2.5 Generating Points Uniformly at Random from a Ball 22

2.6 Gaussians in High Dimension 23

2.7 Random Projection and Johnson-Lindenstrauss Lemma 25

2.8 Separating Gaussians 27

2.9 Fitting a Spherical Gaussian to Data 29

2.10 Bibliographic Notes 31

2.11 Exercises 32

3 Best-Fit Subspaces and Singular Value Decomposition (SVD) 40

3.1 Introduction 40

3.2 Preliminaries 41

3.3 Singular Vectors 42

3.4 Singular Value Decomposition (SVD) 45

3.5 Best Rank-*k* Approximations 47

3.6 Left Singular Vectors 48

3.7 Power Method for Singular Value Decomposition 51

3.7.1 A Faster Method 51

3.8 Singular Vectors and Eigenvectors 54

3.9 Applications of Singular Value Decomposition 54

3.9.1 Centering Data 54

3.9.2 Principal Component Analysis 56
3.9.3 Clustering a Mixture of Spherical Gaussians 56
3.9.4 Ranking Documents and Web Pages 62
3.9.5 An Application of SVD to a Discrete Optimization Problem 63
3.10 Bibliographic Notes 65
3.11 Exercises 67

4 Random Walks and Markov Chains 76

4.1 Stationary Distribution 80
4.2 Markov Chain Monte Carlo 81
4.2.1 Metropolis-Hasting Algorithm 83
4.2.2 Gibbs Sampling 84
4.3 Areas and Volumes 86
4.4 Convergence of Random Walks on Undirected Graphs 88
4.4.1 Using Normalized Conductance to Prove Convergence 94
4.5 Electrical Networks and Random Walks 97
4.6 Random Walks on Undirected Graphs with Unit Edge Weights 102
4.7 Random Walks in Euclidean Space 109
4.8 The Web as a Markov Chain 112
4.9 Bibliographic Notes 116
4.10 Exercises 118

5 Machine Learning 129

5.1 Introduction 129
5.2 The Perceptron algorithm 130
5.3 Kernel Functions 132
5.4 Generalizing to New Data 134
5.5 Overfitting and Uniform Convergence 135
5.6 Illustrative Examples and Occam's Razor 138
5.6.1 Learning Disjunctions 138
5.6.2 Occam's Razor 139
5.6.3 Application: Learning Decision Trees 140
5.7 Regularization: Penalizing Complexity 141
5.8 Online Learning 141
5.8.1 An Example: Learning Disjunctions 142

5.8.2 The Halving Algorithm 143
5.8.3 The Perceptron Algorithm 143
5.8.4 Extensions: Inseparable Data and Hinge Loss 145
5.9 Online to Batch Conversion 146
5.10 Support-Vector Machines 147
5.11 VC-Dimension 148
5.11.1 Definitions and Key Theorems 149
5.11.2 Examples: VC-Dimension and Growth Function 151
5.11.3 Proof of Main Theorems 153
5.11.4 VC-Dimension of Combinations of Concepts 156
5.11.5 Other Measures of Complexity 156
5.12 Strong and Weak Learning - Boosting 157
5.13 Stochastic Gradient Descent 160
5.14 Combining (Sleeping) Expert Advice 162
5.15 Deep Learning 164
5.15.1 Generative Adversarial Networks (GANs) 170
5.16 Further Current Directions 171
5.16.1 Semi-Supervised Learning 171
5.16.2 Active Learning 174
5.16.3 Multi-Task Learning 174
5.17 Bibliographic Notes 175
5.18 Exercises 176

6 Algorithms for Massive Data Problems: Streaming, Sketching, and Sampling 181

6.1 Introduction 181
6.2 Frequency Moments of Data Streams 182
6.2.1 Number of Distinct Elements in a Data Stream 183
6.2.2 Number of Occurrences of a Given Element. 186
6.2.3 Frequent Elements 187
6.2.4 The Second Moment 189
6.3 Matrix Algorithms using Sampling 192
6.3.1 Matrix Multiplication using Sampling 193
6.3.2 Implementing Length Squared Sampling in Two Passes 197
6.3.3 Sketch of a Large Matrix 197
6.4 Sketches of Documents 201
6.5 Bibliographic Notes 203
6.6 Exercises 204

7 Clustering **208**

7.1 Introduction 208
7.1.1 Preliminaries 208
7.1.2 Two General Assumptions on the Form of Clusters 209
7.1.3 Spectral Clustering 211
7.2 k-Means Clustering 211
7.2.1 A Maximum-Likelihood Motivation 211
7.2.2 Structural Properties of the k-Means Objective 212
7.2.3 Lloyd's Algorithm 213
7.2.4 Ward's Algorithm 215
7.2.5 k-Means Clustering on the Line 215
7.3 k-Center Clustering 215
7.4 Finding Low-Error Clusterings 216
7.5 Spectral Clustering 216
7.5.1 Why Project? . . . . . 216 7.5.2 The Algorithm 218
7.5.3 Means Separated by $\Omega(1)$ Standard Deviations 219
7.5.4 Laplacians 221
7.5.5 Local spectral clustering 221
7.6 Approximation Stability 224
7.6.1 The Conceptual Idea 224
7.6.2 Making this Formal 224
7.6.3 Algorithm and Analysis 225
7.7 High-Density Clusters 227
7.7.1 Single Linkage 227
7.7.2 Robust Linkage 228
7.8 Kernel Methods 228
7.9 Recursive Clustering based on Sparse Cuts 229
7.10 Dense Submatrices and Communities 230
7.11 Community Finding and Graph Partitioning 233
7.12 Spectral clustering applied to social networks 236
7.13 Bibliographic Notes 239
7.14 Exercises 240

8 Random Graphs **245**

8.1 The $G(n,p)$ Model 245
8.1.1 Degree Distribution 246
8.1.2 Existence of Triangles in $G(n,d/n)$ 250
8.2 Phase Transitions 252

8.3 Giant Component 261
8.3.1 Existence of a giant component 261
8.3.2 No other large components 263
8.3.3 The case of $p < 1/n$ 264
8.4 Cycles and Full Connectivity 265
8.4.1 Emergence of Cycles 265
8.4.2 Full Connectivity 266
8.4.3 Threshold for $O(\ln n)$ Diameter 268
8.5 Phase Transitions for Increasing Properties 270
8.6 Branching Processes 272
8.7 CNF-SAT 277
8.7.1 SAT-solvers in practice 278
8.7.2 Phase Transitions for CNF-SAT 279
8.8 Nonuniform Models of Random Graphs 284
8.8.1 Giant Component in Graphs with Given Degree Distribution 285
8.9 Growth Models 286
8.9.1 Growth Model Without Preferential Attachment 287
8.9.2 Growth Model With Preferential Attachment 293
8.10 Small World Graphs 294
8.11 Bibliographic Notes 299
8.12 Exercises 301

9 Topic Models, Nonnegative Matrix Factorization, Hidden Markov Models, and Graphical Models 310

9.1 Topic Models 310
9.2 An Idealized Model 313
9.3 Nonnegative Matrix Factorization - NMF 315
9.4 NMF with Anchor Terms 317
9.5 Hard and Soft Clustering 318
9.6 The Latent Dirichlet Allocation Model for Topic Modeling 320
9.7 The Dominant Admixture Model 322
9.8 Formal Assumptions 324
9.9 Finding the Term-Topic Matrix 327
9.10 Hidden Markov Models 332
9.11 Graphical Models and Belief Propagation 337
9.12 Bayesian or Belief Networks 338
9.13 Markov Random Fields 339
9.14 Factor Graphs 340
9.15 Tree Algorithms 341

9.16 Message Passing in General Graphs 342
9.17 Graphs with a Single Cycle 344
9.18 Belief Update in Networks with a Single Loop 346
9.19 Maximum Weight Matching 347
9.20 Warning Propagation 351
9.21 Correlation Between Variables 351
9.22 Bibliographic Notes 355
9.23 Exercises 357

10 Other Topics **360**
10.1 Ranking and Social Choice 360
10.1.1 Randomization 362
10.1.2 Examples 363
10.2 Compressed Sensing and Sparse Vectors 364
10.2.1 Unique Reconstruction of a Sparse Vector 365
10.2.2 Efficiently Finding the Unique Sparse Solution 366
10.3 Applications 368
10.3.1 Biological 368
10.3.2 Low Rank Matrices 369
10.4 An Uncertainty Principle 370
10.4.1 Sparse Vector in Some Coordinate Basis . . . . . 370 10.4.2 A Representation Cannot be Sparse in Both Time and Frequency Domains 371
10.5 Gradient 373
10.6 Linear Programming 375
10.6.1 The Ellipsoid Algorithm 375
10.7 Integer Optimization 377
10.8 Semi-Definite Programming 378
10.9 Bibliographic Notes 380
10.10Exercises 381
11 Wavelets **385**
11.1 Dilation 385
11.2 The Haar Wavelet 386
11.3 Wavelet Systems 390
11.4 Solving the Dilation Equation 390
11.5 Conditions on the Dilation Equation 392
11.6 Derivation of the Wavelets from the Scaling Function 394
11.7 Sufficient Conditions for the Wavelets to be Orthogonal 398
11.8 Expressing a Function in Terms of Wavelets 401

11.9 Designing a Wavelet System 402
11.10Applications 402
11.11 Bibliographic Notes 402
11.12 Exercises 403

12 Appendix **406**
12.1 Definitions and Notation 406
12.2 Asymptotic Notation 406
12.3 Useful Relations 408
12.4 Useful Inequalities 413
12.5 Probability 420
12.5.1 Sample Space, Events, and Independence 420
12.5.2 Linearity of Expectation 421
12.5.3 Union Bound 422
12.5.4 Indicator Variables 422
12.5.5 Variance 422
12.5.6 Variance of the Sum of Independent Random Variables 423
12.5.7 Median 423
12.5.8 The Central Limit Theorem 423
12.5.9 Probability Distributions 424
12. 5.10Bayes Rule and Estimators 428
12.6 Bounds on Tail Probability 430
12.6.1 Chernoff Bounds 430
12.6.2 More General Tail Bounds 433
12.7 Applications of the Tail Bound 436
12.8 Eigenvalues and Eigenvectors 437
12.8.1 Symmetric Matrices 439
12.8.2 Relationship between SVD and Eigen Decomposition 441
12.8.3 Extremal Properties of Eigenvalues 441
12.8.4 Eigenvalues of the Sum of Two Symmetric Matrices 443
12.8.5 Norms 445
12.8.6 Important Norms and Their Properties 446
12.8.7 Additional Linear Algebra 448
12.8.8 Distance between subspaces 450
12.8.9 Positive semidefinite matrix 451
12.9 Generating Functions 451

12.9.1 Generating Functions for Sequences Defined by Recurrence Relationships 452
12.9.2 The Exponential Generating Function and the Moment Generating Function 454
12.10 Miscellaneous 456
12.10.1 Lagrange multipliers 456
12.10.2 Finite Fields 457
12.10.3 Application of Mean Value Theorem 457
12.10.4 Sperner's Lemma 459
12.10.5 Pru¨fer 459
12.11 Exercises 460

Index **466**

1 Introduction

Computer science as an academic discipline began in the 1960's. Emphasis was on programming languages, compilers, operating systems, and the mathematical theory that supported these areas. Courses in theoretical computer science covered finite automata, regular expressions, context-free languages, and computability. In the 1970's, the study of algorithms was added as an important component of theory. The emphasis was on making computers useful. Today, a fundamental change is taking place and the focus is more on a wealth of applications. There are many reasons for this change. The merging of computing and communications has played an important role. The enhanced ability to observe, collect, and store data in the natural sciences, in commerce, and in other fields calls for a change in our understanding of data and how to handle it in the modern setting. The emergence of the web and social networks as central aspects of daily life presents both opportunities and challenges for theory.

While traditional areas of computer science remain highly important, increasingly researchers of the future will be involved with using computers to understand and extract usable information from massive data arising in applications, not just how to make computers useful on specific well-defined problems. With this in mind we have written this book to cover the theory we expect to be useful in the next 40 years, just as an understanding of automata theory, algorithms, and related topics gave students an advantage in the last 40 years. One of the major changes is an increase in emphasis on probability, statistics, and numerical methods.

Early drafts of the book have been used for both undergraduate and graduate courses. Background material needed for an undergraduate course has been put in the appendix. For this reason, the appendix has homework problems.

Modern data in diverse fields such as information processing, search, and machine learning is often advantageously represented as vectors with a large number of components. The vector representation is not just a book-keeping device to store many fields of a record. Indeed, the two salient aspects of vectors: geometric (length, dot products, orthogonality etc.) and linear algebraic (independence, rank, singular values etc.) turn out to be relevant and useful. Chapters 2 and 3 lay the foundations of geometry and linear algebra respectively. More specifically, our intuition from two or three dimensional space can be surprisingly off the mark when it comes to high dimensions. Chapter 2 works out the fundamentals needed to understand the differences. The emphasis of the chapter, as well as the book in general, is to get across the intellectual ideas and the mathematical foundations rather than focus on particular applications, some of which are briefly described. Chapter 3 focuses on singular value decomposition (SVD) a central tool to deal with matrix data. We give a from-first-principles description of the mathematics and algorithms for SVD. Applications of singular value decomposition include principal component analysis, a widely used technique which we touch upon, as

well as modern applications to statistical mixtures of probability densities, discrete optimization, etc., which are described in more detail.

Exploring large structures like the web or the space of configurations of a large system with deterministic methods can be prohibitively expensive. Random walks (also called Markov Chains) turn out often to be more efficient as well as illuminative. The stationary distributions of such walks are important for applications ranging from web search to the simulation of physical systems. The underlying mathematical theory of such random walks, as well as connections to electrical networks, forms the core of Chapter 4 on Markov chains.

One of the surprises of computer science over the last two decades is that some domainindependent methods have been immensely successful in tackling problems from diverse areas. Machine learning is a striking example. Chapter 5 describes the foundations of machine learning, both algorithms for optimizing over given training examples, as well as the theory for understanding when such optimization can be expected to lead to good performance on new, unseen data. This includes important measures such as the Vapnik-Chervonenkis dimension, important algorithms such as the Perceptron Algorithm, stochastic gradient descent, boosting, and deep learning, and important notions such as regularization and overfitting.

The field of algorithms has traditionally assumed that the input data to a problem is presented in random access memory, which the algorithm can repeatedly access. This is not feasible for problems involving enormous amounts of data. The streaming model and other models have been formulated to reflect this. In this setting, sampling plays a crucial role and, indeed, we have to sample on the fly. In Chapter 6 we study how to draw good samples efficiently and how to estimate statistical and linear algebra quantities, with such samples.

While Chapter 5 focuses on supervised learning, where one learns from labeled training data, the problem of unsupervised learning, or learning from unlabeled data, is equally important. A central topic in unsupervised learning is clustering, discussed in Chapter 7. Clustering refers to the problem of partitioning data into groups of similar objects. After describing some of the basic methods for clustering, such as the k-means algorithm, Chapter 7 focuses on modern developments in understanding these, as well as newer algorithms and general frameworks for analyzing different kinds of clustering problems.

Central to our understanding of large structures, like the web and social networks, is building models to capture essential properties of these structures. The simplest model is that of a random graph formulated by Erd¨os and Renyi, which we study in detail in Chapter 8, proving that certain global phenomena, like a giant connected component, arise

in such structures with only local choices. We also describe other models of random graphs.

Chapter 9 focuses on linear-algebraic problems of making sense from data, in particular topic modeling and non-negative matrix factorization. In addition to discussing well-known models, we also describe some current research on models and algorithms with provable guarantees on learning error and time. This is followed by graphical models and belief propagation.

Chapter 10 discusses ranking and social choice as well as problems of sparse representations such as compressed sensing. Additionally, Chapter 10 includes a brief discussion of linear programming and semidefinite programming. Wavelets, which are an important method for representing signals across a wide range of applications, are discussed in Chapter 11 along with some of their fundamental mathematical properties. The appendix includes a range of background material.

A word about notation in the book. To help the student, we have adopted certain notations, and with a few exceptions, adhered to them. We use lower case letters for scalar variables and functions, bold face lower case for vectors, and upper case letters for matrices. Lower case near the beginning of the alphabet tend to be constants, in the middle of the alphabet, such as i, j, and k, are indices in summations, n and m for integer sizes, and x, y and z for variables. If A is a matrix its elements are a_{ij} and its rows are $\mathbf{a_i}$. If $\mathbf{a_i}$ is a vector its coordinates are a_{ij}. Where the literature traditionally uses a symbol for a quantity, we also used that symbol, even if it meant abandoning our convention. If we have a set of points in some vector space, and work with a subspace, we use n for the number of points, d for the dimension of the space, and k for the dimension of the subspace.

The term "almost surely" means with probability tending to one. We use $\ln n$ for the natural logarithm and $\log n$ for the base two logarithm. If we want base ten, we will use $\log_{10}$. To simplify notation and to make it easier to read we use $E^2(1-x)$ for $\left(E(1-x)\right)^2$ and $E(1-x)^2$ for $E\left((1-x)^2\right)$. When we say "randomly select" some number of points from a given probability distribution, independence is always assumed unless otherwise stated.

2 High-Dimensional Space

2.1 Introduction

High dimensional data has become very important. However, high dimensional space is very different from the two and three dimensional spaces we are familiar with. Generate n points at random in d-dimensions where each coordinate is a zero mean, unit variance Gaussian. For sufficiently large d, with high probability the distances between all pairs of points will be essentially the same. Also the volume of the unit ball in d-dimensions, the

set of all points **x** such that $|\mathbf{x}| \leq 1$, goes to zero as the dimension goes to infinity. The volume of a high dimensional unit ball is concentrated near its surface and is also concentrated at its equator. These properties have important consequences which we will consider.

2.2 The Law of Large Numbers

If one generates random points in *d*-dimensional space using a Gaussian to generate coordinates, the distance between all pairs of points will be essentially the same when *d* is large. The reason is that the square of the distance between two points **y** and **z**,

$$|\mathbf{y} - \mathbf{z}|^2 = \sum_{i=1}^{d} (y_i - z_i)^2,$$

can be viewed as the sum of *d* independent samples of a random variable *x* that is distributed as the squared difference of two Gaussians. In particular, we are summing independent samples $x_i = (y_i - z_i)^2$ of a random variable *x* of bounded variance. In such a case, a general bound known as the Law of Large Numbers states that with high probability, the average of the samples will be close to the expectation of the random variable. This in turn implies that with high probability, the sum is close to the sum's expectation.

Specifically, the Law of Large Numbers states that

$$\text{Prob}\left(\left|\frac{x_1 + x_2 + \cdots + x_n}{n} - E(x)\right| \geq \epsilon\right) \leq \frac{Var(x)}{n\epsilon^2}. \tag{2.1}$$

The larger the variance of the random variable, the greater the probability that the error will exceed . Thus the variance of *x* is in the numerator. The number of samples *n* is in the denominator since the more values that are averaged, the smaller the probability that the difference will exceed . Similarly the larger is, the smaller the probability that the difference will exceed and hence is in the denominator. Notice that squaring makes the fraction a dimensionless quantity.

We use two inequalities to prove the Law of Large Numbers. The first is Markov's inequality that states that the probability that a nonnegative random variable exceeds *a* is bounded by the expected value of the variable divided by *a*.

Theorem 2.1 (Markov's inequality) *Let x be a nonnegative random variable. Then for a* > 0,

$$Prob(x \geq a) \leq \frac{E(x)}{a}.$$

Proof: For a continuous nonnegative random variable *x* with probability density *p*,

$$E(x) = \int_0^{\infty} xp(x)dx = \int_0^{a} xp(x)dx + \int_a^{\infty} xp(x)dx$$

$$\geq \int_a^{\infty} xp(x)dx \geq a \int_a^{\infty} p(x)dx = a\text{Prob}(x \geq a).$$

Thus, Prob($x \geq a) \leq \frac{E(x)}{a}$. ∎

The same proof works for discrete random variables with sums instead of integrals.

Corollary 2.2 *Prob*$\left(x \geq bE(x)\right) \leq \frac{1}{b}$

Markov's inequality bounds the tail of a distribution using only information about the mean. A tighter bound can be obtained by also using the variance of the random variable.

Theorem 2.3 (Chebyshev's inequality) *Let x be a random variable. Then for c* > 0*,*

$$Prob\Big(|x - E(x)| \geq c\Big) \leq \frac{Var(x)}{c^2}.$$

Proof: Prob$|x - E(x)| \geq c)$= Prob$|x - E(x)|^2 \geq c^2)$. Let $y = |x - E(x)|^2$. Note that y is a nonnegative random variable and $E(y) = Var(x)$, so Markov's inequality can be applied giving:

$$\text{Prob}(|x - E(x)| \geq c) = \text{Prob}\,|x - E(x)|^2 \geq c^2) \leq \frac{E(|x - E(x)|^2)}{c^2} = \frac{Var(x)}{c^2}.$$

∎

The Law of Large Numbers follows from Chebyshev's inequality together with facts about independent random variables. Recall that:

$$E(x + y) = E(x) + E(y),\ Var(x - c) = Var(x),\ Var(cx) = c^2 Var(x).$$

Also, if x and y are independent, then $E(xy) = E(x)E(y)$. These facts imply that if x and y are independent then $Var(x + y) = Var(x) + Var(y)$, which is seen as follows:

$$\begin{aligned} Var(x+y) &= E(x+y)^2 - E^2(x+y) \\ &= E(x^2 + 2xy + y^2) - \left(E^2(x) + 2E(x)E(y) + E^2(y)\right) \\ &= E(x^2) - E^2(x) + E(y^2) - E^2(y) = Var(x) + Var(y), \end{aligned}$$

where we used independence to replace $E(2xy)$ with $2E(x)E(y)$.

Theorem 2.4 (Law of Large Numbers) *Let $x_1,x_2,...,x_n$ be n independent samples of a random variable x. Then*

$$Prob\left(\left|\frac{x_1+x_2+\cdots+x_n}{n}-E(x)\right|\geq\epsilon\right)\leq\frac{Var(x)}{n\epsilon^2}$$

Proof: By Chebychev's inequality

$$\begin{aligned}\left(\left|\frac{x_1+x_2+\cdots+x_n}{n}-E(x)\right|\geq\epsilon\right)&\leq\frac{Var\left(\frac{x_1+x_2+\cdots+x_n}{n}\right)}{\epsilon^2}\\&=\frac{1}{n^2\epsilon^2}Var(x_1+x_2+\cdots+x_n)\\&=\frac{1}{n^2\epsilon^2}\left(Var(x_1)+Var(x_2)+\cdots+Var(x_n)\right)\\&=\frac{Var(x)}{n\epsilon^2}.\end{aligned}$$

Prob ∎

The Law of Large Numbers is quite general, applying to any random variable *x* of finite variance. Later we will look at tighter concentration bounds for spherical Gaussians and sums of 0-1 valued random variables.

One observation worth making about the Law of Large Numbers is that the size of the universe does not enter into the bound. For instance, if you want to know what fraction of the population of a country prefers tea to coffee, then the number *n* of people you need to sample in order to have at most a δ chance that your estimate is off by more than depends only on and δ and not on the population of the country.

As an application of the Law of Large Numbers, let **z** be a *d*-dimensional random point whose coordinates are each selected from a zero mean, $\frac{1}{2\pi}$ variance Gaussian. We set the variance to $\frac{1}{2\pi}$so the Gaussian probability density equals one at the origin and is bounded below throughout the unit ball by a constant.[2] By the Law of Large Numbers, the square of the distance of **z** to the origin will be $\Theta(d)$ with high probability. In particular, there is vanishingly small probability that such a random point **z** would lie in the unit ball. This implies that the integral of the probability density over the unit ball must be vanishingly small. On the other hand, the probability density in the unit ball is bounded below by a constant. We thus conclude that the unit ball must have vanishingly small volume.

Similarly if we draw two points **y** and **z** from a *d*-dimensional Gaussian with unit variance in each direction, then $|\mathbf{y}|^2 \approx d$ and $|\mathbf{z}|^2 \approx d$. Since for all *i,*

[2] If we instead used variance 1, then the density at the origin would be a decreasing function of *d*, namely $(\frac{1}{2\pi})^{d/2}$, making this argument more complicated.

$$E(y_i - z_i)^2 = E(y_i^2) + E(z_i^2) - 2E(y_i z_i) = Var(y_i) + Var(z_i) + 2E(y_i)E(z_i) = 2,$$

$|\mathbf{y}-\mathbf{z}|^2 = \sum_{i=1}^{d}(y_i-z_i)^2 \approx 2d$. Thus by the Pythagorean theorem, the random d-dimensional **y** and **z** must be approximately orthogonal. This implies that if we scale these random points to be unit length and call **y** the North Pole, much of the surface area of the unit ball must lie near the equator. We will formalize these and related arguments in subsequent sections.

We now state a general theorem on probability tail bounds for a sum of independent random variables. Tail bounds for sums of Bernoulli, squared Gaussian and Power Law distributed random variables can all be derived from this. The table in Figure 2.1 summarizes some of the results.

Theorem 2.5 (Master Tail Bounds Theorem) *Let $x = x_1 + x_2 + \cdots + x_n$, where $x_1, x_2, \ldots, x_n$ are mutually independent random variables with zero mean and variance at most σ^2. Let $0 \le a \le \sqrt{2n\sigma^2}$. Assume that $|E(x_i^s)| \le \sigma^2 s!$ for $s = 3, 4, \ldots, \lfloor a^2/4n\sigma^2 \rfloor$. Then,*

$$Prob(|x| \ge a) \le 3e^{-a^2/(12n\sigma^2)}.$$

The proof of Theorem 2.5 is elementary. A slightly more general version, Theorem 12.5, is given in the appendix. For a brief intuition of the proof, consider applying Markov's inequality to the random variable x^r where r is a large even number. Since r is even, x^r is nonnegative, and thus $\text{Prob}(|x| \ge a) = \text{Prob}(x^r \ge a^r) \le E(x^r)/a^r$. If $E(x^r)$ is not too large, we will get a good bound. To compute $E(x^r)$, write $E(x)$ as $E(x_1 + \ldots + x_n)^r$ and expand the polynomial into a sum of terms. Use the fact that by independence $E(x_i^{r_i} x_j^{r_j}) = E(x_i^{r_i})E(x_j^{r_j})$ to get a collection of simpler expectations that can be bounded using our assumption that $|E(x_i^s)| \le \sigma^2 s!$. For the full proof, see the appendix.

2.3 The Geometry of High Dimensions

An important property of high-dimensional objects is that most of their volume is near the surface. Consider any object A in R^d. Now shrink A by a small amount ϵ to produce a new object $(1-\epsilon)A = \{(1-\epsilon)x | x \in A\}$. Then the following equality holds:

$$\text{volume}\big((1-\epsilon)A\big) = (1-\epsilon)^d \text{volume}(A).$$

	Condition	Tail bound

Markov	$x \geq 0$	Prob$(x \geq a) \leq \frac{E(x)}{a}$
Chebychev	Any x	Prob $\left(\lvert x - E(x)\rvert \geq a\right) \leq \frac{\mathrm{Var}(x)}{a^2}$
Chernoff	$x = x_1 + x_2 + \cdots + x_n$ $x_i \in [0,1]$ i.i.d. Bernoulli;	Prob$(\lvert x - E(x)\rvert \geq \varepsilon E(x))$ $\leq 3e^{-c\varepsilon^2 E(x)}$
Higher Moments	r positive even integer	Prob$(\lvert x\rvert \geq a) \leq E(x^r)/a^r$
Gaussian Annulus	$x = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ $x_i \sim N(0,1); \beta \leq \sqrt{n}$ indep.	Prob$(\lvert x - \sqrt{n}\rvert \geq \beta) \leq 3e^{-c\beta^2}$
Power Law for x_i; order $k \geq 4$	$x = x_1 + x_2 + \ldots + x_n$ x_i i.i.d ; $\varepsilon \leq 1/k^2$	Prob $\left(\lvert x - E(x)\rvert \geq \varepsilon E(x)\right)$ $\leq (4/\varepsilon^2 kn)^{(k-3)/2}$

Figure 2.1: Table of Tail Bounds. The Higher Moments bound is obtained by applying Markov to x^r. The Chernoff, Gaussian Annulus, and Power Law bounds follow from Theorem 2.5 which is proved in the appendix.

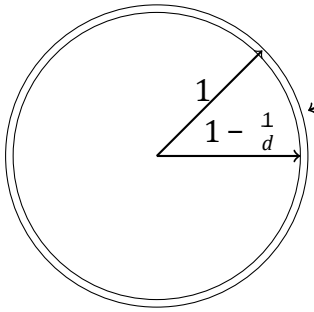
To see that this is true, partition A into infinitesimal cubes. Then, $(1-\varepsilon)A$ is the union of a set of cubes obtained by shrinking the cubes in A by a factor of $1-\varepsilon$. When we shrink each of the $2d$ sides of a d-dimensional cube by a factor f, its volume shrinks by a factor of f^d. Using the fact that $1 - x \leq e^{-x}$, for any object A in R^d we have:

$$\frac{\text{volume}\left((1-\epsilon)A\right)}{\text{volume}(A)} = (1-\epsilon)^d \leq e^{-\epsilon d}.$$

Fixing ϵ and letting $d \to \infty$, the above quantity rapidly approaches zero. This means that nearly all of the volume of A must be in the portion of A that does not belong to the region $(1-\epsilon)A$.

Let S denote the unit ball in d dimensions, that is, the set of points within distance one of the origin. An immediate implication of the above observation is that at least a $1 - e^{-\epsilon d}$ fraction of the volume of the unit ball is concentrated in $S \setminus (1-\epsilon)S$, namely in a small annulus of width ϵ at the boundary. In particular, most of the volume of the d-dimensional

unit ball is contained $O(1/d)$ near the of radius r, then the **Figure 2.2:** Most of dimensional ball of annulus of width



Annulusof width $\frac{1}{d}$ in an annulus of width boundary. If the ball is annulus width is $O\left(\frac{r}{d}\right)$. the volume of the d-radius r is contained in an $O(r/d)$ near the boundary.

2.4 Properties of the Unit Ball

We now focus more specifically on properties of the unit ball in d-dimensional space. We just saw that most of its volume is concentrated in a small annulus of width $O(1/d)$ near the boundary. Next we will show that in the limit as d goes to infinity, the volume of the ball goes to zero. This result can be proven in several ways. Here we use integration.

2.4.1 Volume of the Unit Ball

To calculate the volume $V(d)$ of the unit ball in R^d, one can integrate in either Cartesian or polar coordinates. In Cartesian coordinates the volume is given by

$$V(d) = \int_{x_1=-1}^{x_1=1} \int_{x_2=-\sqrt{1-x_1^2}}^{x_2=\sqrt{1-x_1^2}} \cdots \int_{x_d=-\sqrt{1-x_1^2-\cdots-x_{d-1}^2}}^{x_d=\sqrt{1-x_1^2-\cdots-x_{d-1}^2}} dx_d \cdots dx_2 dx_1 .$$

Since the limits of the integrals are complicated, it is easier to integrate using polar coordinates. In polar coordinates, $V(d)$ is given by

$$V(d) = \int_{S^d} \int_{r=0}^{1} r^{d-1} dr d\Omega.$$

Since the variables Ω and r do not interact,

$$V(d) = \int_{S^d} d\Omega \int_{r=0}^{1} r^{d-1} dr = \frac{1}{d} \int_{S^d} d\Omega = \frac{A(d)}{d}$$

where $A(d)$ is the surface area of the d-dimensional unit ball. For instance, for $d = 3$ the surface area is 4π and the volume is $\frac{4}{3}\pi$. The question remains, how to determine the surface area $A(d) = \int_{S^d} d\Omega$ for general d.

Consider a different integral

$$I(d) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{-\left(x_1^2 + x_2^2 + \cdots x_d^2\right)} dx_d \cdots dx_2 dx_1 .$$

Including the exponential allows integration to infinity rather than stopping at the surface of the sphere. Thus, $I(d)$ can be computed by integrating in both Cartesian and polar coordinates. Integrating in polar coordinates will relate $I(d)$ to the surface area $A(d)$. Equating the two results for $I(d)$ allows one to solve for $A(d)$.

First, calculate $I(d)$ by integration in Cartesian coordinates.

$$I(d) = \left[\int_{-\infty}^{\infty} e^{-x^2} dx\right]^d = \left(\sqrt{\pi}\right)^d = \pi^{\frac{d}{2}} .$$

Here, we have used the fact that $\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$. For a proof of this, see Section 12.3 of the appendix. Next, calculate $I(d)$ by integrating in polar coordinates. The volume of the differential element is $r^{d-1} d\Omega dr$. Thus,

$$I(d) = \int_{S^d} d\Omega \int_0^{\infty} e^{-r^2} r^{d-1} dr.$$

The integral $\int_{S^d} d\Omega$ is the integral over the entire solid angle and gives the surface area, $A(d)$, of a unit sphere. Thus, $I(d) = A(d) \int_0^{\infty} e^{-r^2} r^{d-1} dr$. Evaluating the remaining integral gives

$$\int_0^{\infty} e^{-r^2} r^{d-1} dr = \int_0^{\infty} e^{-t} t^{\frac{d-1}{2}} \left(\tfrac{1}{2} t^{-\frac{1}{2}} dt\right) = \frac{1}{2} \int_0^{\infty} e^{-t} t^{\frac{d}{2} - 1} dt = \frac{1}{2} \Gamma\left(\frac{d}{2}\right)$$

and hence, $I(d) = A(d) \frac{1}{2} \Gamma\left(\frac{d}{2}\right)$ where the Gamma function $\Gamma(x)$ is a generalization of the factorial function for noninteger values of x. $\Gamma(x) = (x-1)\Gamma(x-1)$, $\Gamma(1) = \Gamma(2) = 1$, and $\Gamma\left(-\frac{1}{2}\right) = \sqrt{\pi}$. For integer x, $\Gamma(x) = (x-1)!$.

Combining $I(d) = \pi^{\frac{d}{2}}$ with $I(d) = A(d) \frac{1}{2} \Gamma\left(\frac{d}{2}\right)$ yields

$$A(d) = \frac{\pi^{\frac{d}{2}}}{\frac{1}{2}\Gamma\left(\frac{d}{2}\right)}$$

establishing the following lemma.

Lemma 2.6 *The surface area $A(d)$ and the volume $V(d)$ of a unit-radius ball in d dimensions are given by*

$$A(d) = \frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})} \quad \text{and} \quad V(d) = \frac{2\pi^{\frac{d}{2}}}{d\,\Gamma(\frac{d}{2})}.$$

To check the formula for the volume of a unit ball, note that $V(2) = \pi$ and $V(3) = \frac{2}{3}\frac{\pi^{\frac{3}{2}}}{\Gamma(\frac{3}{2})} = \frac{4}{3}\pi$, which are the correct volumes for the unit balls in two and three dimensions. To check the formula for the surface area of a unit ball, note that $A(2) = 2\pi$ and $A(3) = \frac{2\pi^{\frac{3}{2}}}{\frac{1}{2}\sqrt{\pi}} = 4\pi$, which are the correct surface areas for the unit ball in two and three dimensions. Note that $\pi^{\frac{d}{2}}$ is an exponential in $\frac{d}{2}$ and $\Gamma\left(\frac{d}{2}\right)$ grows as the factorial of $\frac{d}{2}$. This implies that $\lim_{d\to\infty} V(d) = 0$, as claimed.

2.4.2 Volume Near the Equator

An interesting fact about the unit ball in high dimensions is that most of its volume is concentrated near its "equator". In particular, for any unit-length vector **v** defining "north", most of the volume of the unit ball lies in the thin slab of points whose dot-product with **v** has magnitude $O(1/\sqrt{d})$. To show this fact, it suffices by symmetry to fix **v** to be the first coordinate vector. That is, we will show that most of the volume of the unit ball has $|x_1| = O(1/\sqrt{d})$. Using this fact, we will show that two random points in the unit ball are with high probability nearly orthogonal, and also give an alternative proof from the one in Section 2.4.1 that the volume of the unit ball goes to zero as $d \to \infty$.

Theorem 2.7 *For $c \geq 1$ and $d \geq 3$, at least a $1 - \frac{2}{c}e^{-c^2/2}$ fraction of the volume of the d-dimensional unit ball has $|x_1| \leq \frac{c}{\sqrt{d-1}}$.*

Proof: By symmetry we just need to prove that at most a $\frac{2}{c}e^{-c^2/2}$ fraction of the half of the ball with $x_1 \geq 0$ has $x_1 \geq \frac{c}{\sqrt{d-1}}$. Let A denote the portion of the ball with $x_1 \geq \frac{c}{\sqrt{d-1}}$ and let H denote the upper hemisphere. We will then show that the ratio of the volume of A to the volume of H goes to zero by calculating an upper bound on volume(A) and a lower bound on volume(H) and proving that

$$\frac{\text{volume}(A)}{\text{volume}(H)} \leq \frac{\text{upper bound volume}(A)}{\text{lower bound volume}(H)} = \frac{2}{c}e^{-\frac{c^2}{2}}.$$

To calculate the volume of A, integrate an incremental volume that is a disk of width dx_1 and whose face is a ball of dimension $d-1$ and radius $\sqrt{1-x_1^2}$. The surface area of the disk is $(1-x_1^2)^{\frac{d-1}{2}}V(d-1)$ and the volume above the slice is

$$\text{volume}(A) = \int_{\frac{c}{\sqrt{d-1}}}^{1} (1-x_1^2)^{\frac{d-1}{2}} V(d-1)dx_1$$

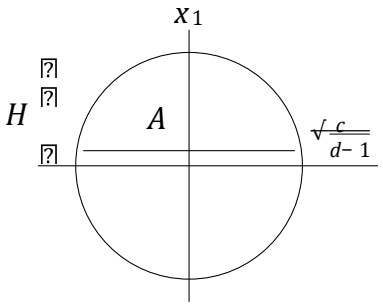


Figure 2.3: Most of the volume of the upper hemisphere of the d-dimensional ball is below the plane $x_1 = \frac{c}{\sqrt{d-1}}$.

To get an upper bound on the above integral, use $1 - x \le e^{-x}$ and integrate to infinity. To integrate, insert $\frac{x_1\sqrt{d-1}}{c}$, which is greater than one in the range of integration, into the integral. Then

$$\text{volume}(A) \le \int_{\frac{c}{\sqrt{d-1}}}^{\infty} \frac{x_1\sqrt{d-1}}{c} e^{-\frac{d-1}{2}x_1^2} V(d-1)dx_1 = V(d-1)\frac{\sqrt{d-1}}{c}\int_{\frac{c}{\sqrt{d-1}}}^{\infty} x_1 e^{-\frac{d-1}{2}x_1^2}dx_1$$

Now

$$\int_{\frac{c}{\sqrt{d-1}}}^{\infty} x_1 e^{-\frac{d-1}{2}x_1^2}dx_1 = -\frac{1}{d-1}e^{-\frac{d-1}{2}x_1^2}\Big|_{\frac{c}{\sqrt{(d-1)}}}^{\infty} = \frac{1}{d-1}e^{-\frac{c^2}{2}}$$

Thus, an upper bound on volume(A) is $\frac{V(d-1)}{c\sqrt{d-1}}e^{-\frac{c^2}{2}}$.

The volume of the hemisphere below the plane $x_1 = \frac{1}{\sqrt{d-1}}$ is a lower bound on the entire volume of the upper hemisphere and this volume is at least that of a cylinder of height $\frac{1}{\sqrt{d-1}}$ and radius $\sqrt{1-\frac{1}{d-1}}$. The volume of the cylinder is $V(d-1)(1-\frac{1}{d-1})^{\frac{d-1}{2}}\frac{1}{\sqrt{d-1}}$. Using the fact that $(1-x)^a \ge 1-ax$ for $a \ge 1$, the volume of the cylinder is at least $\frac{V(d-1)}{2\sqrt{d-1}}$ for $d \ge 3$.

Thus,

upper bound above plane

$$\text{ratio} \leq \frac{\frac{V(d-1)}{c\sqrt{d-1}} e^{-\frac{c^2}{2}}}{\text{lower bound total hemisphere}} = \frac{\frac{V(d-1)}{c\sqrt{d-1}} e^{-\frac{c^2}{2}}}{\frac{V(d-1)}{2\sqrt{d-1}}} = \frac{2}{c} e^{-\frac{c^2}{2}}$$

■

One might ask why we computed a lower bound on the total hemisphere since it is one half of the volume of the unit ball which we already know. The reason is that the volume of the upper hemisphere is $\frac{1}{2}V(d)$ and we need a formula with $V(d-1)$ in it to cancel the $V(d-1)$ in the numerator.

Near orthogonality. One immediate implication of the above analysis is that if we draw two points at random from the unit ball, with high probability their vectors will be nearly orthogonal to each other. Specifically, from our previous analysis in Section 2.3, with high probability both will be close to the surface and will have length $1 - O(1/d)$. From our analysis above, if we define the vector in the direction of the first point as "north", with high probability the second will have a projection of only $\pm O(1/\sqrt{d})$ in this direction, and thus their dot-product will be $\pm O(1/\sqrt{d})$. This implies that with high probability, the angle between the two vectors will be $\pi/2 \pm O(1/\sqrt{d})$. In particular, we have the following theorem that states that if we draw n points at random in the unit ball, with high probability all points will be close to unit length and each pair of points will be almost orthogonal.

Theorem 2.8 *Consider drawing n points $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ at random from the unit ball. With probability $1 - O(1/n)$*

1. $|\mathbf{x_i}| \geq 1 - \frac{2\ln n}{d}$ *for all i, and*
2. $|\mathbf{x_i} \cdot \mathbf{x_j}| \leq \frac{\sqrt{6\ln n}}{\sqrt{d-1}}$ *for all $i \neq j$.*

Proof: For the first part, for any fixed i by the analysis of Section 2.3, the probability that $|\mathbf{x_i}| < 1 - \epsilon$ is less than $e^{-\epsilon d}$. Thus

$$\text{Prob}\left(|\mathbf{x_i}| < 1 - \frac{2\ln n}{d}\right) \leq e^{-(\frac{2\ln n}{d})d} = 1/n^2.$$

By the union bound, the probability there exists an i such that $|\mathbf{x_i}| < 1 - \frac{2\ln n}{d}$ is at most $1/n$.

For the second part, Theorem 2.7 states that the probability $|\mathbf{x_i}| > \frac{c}{\sqrt{d-1}}$ is at most $\frac{2}{c}e^{-\frac{c^2}{2}}$. There are $\binom{n}{2}$ pairs i and j and for each such pair if we define $\mathbf{x_i}$ as "north", the probability that the projection of $\mathbf{x_j}$ onto the "north" direction is more than $\frac{\sqrt{6\ln n}}{\sqrt{d-1}}$ is at most $O(e^{-\frac{6\ln n}{2}}) = O(n^{-3})$. Thus, the dot-product condition is violated with probability at most $O\left(\binom{n}{2}n^{-3}\right) = O(1/n)$ as well. ■

Alternative proof that volume goes to zero. Another immediate implication of Theorem 2.7 is that as $d \to \infty$, the volume of the ball approaches zero. Specifically, consider a small box centered at the origin of side length $\frac{2c}{\sqrt{d-1}}$. Using Theorem 2.7, we show

that for $c = 2 \ln d$, this box contains over half of the volume of the ball. On the other hand, the volume of this box clearly goes to zero as d goes to infinity, since its volume is $O((\frac{\ln d}{d-1})^{d/2})$. Thus the volume of the ball goes to zero as well.

By Theorem 2.7 with $c = 2 \ln d$, the fraction of the volume of the ball with $|x_1| \geq \frac{c}{\sqrt{d-1}}$ is at most:

$$\frac{2}{c}e^{-\frac{c^2}{2}} = \frac{1}{\sqrt{\ln d}}e^{-2\ln d} = \frac{1}{d^2\sqrt{\ln d}} < \frac{1}{d^2}.$$

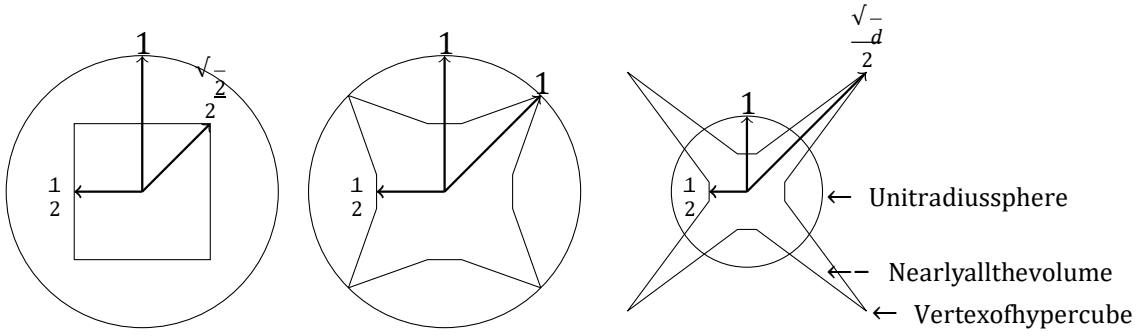


Figure 2.4: Illustration of the relationship between the sphere and the cube in 2, 4, and d-dimensions.

Since this is true for each of the d dimensions, by a union bound at most a $O(\frac{1}{d}) \leq \frac{1}{2}$ fraction of the volume of the ball lies outside the cube, completing the proof.

Discussion. One might wonder how it can be that nearly all the points in the unit ball are very close to the surface and yet at the same time nearly all points are in a box of side-length $O\left(\frac{\ln d}{d-1}\right)$. The answer is to remember that points on the surface of the ball satisfy $x_1^2 + x_2^2 + \ldots + x_d^2 = 1$, so for each coordinate i, a typical value will be $\pm O\left(\frac{1}{\sqrt{d}}\right)$.
In fact, it is often helpful to think of picking a random point on the sphere as very similar to picking a random point of the form $\left(\pm\frac{1}{\sqrt{d}}, \pm\frac{1}{\sqrt{d}}, \pm\frac{1}{\sqrt{d}}, \ldots \pm\frac{1}{\sqrt{d}}\right)$.

2.5 Generating Points Uniformly at Random from a Ball

Consider generating points uniformly at random on the surface of the unit ball. For the 2-dimensional version of generating points on the circumference of a unit-radius circle, independently generate each coordinate uniformly at random from the interval [−1,1]. This produces points distributed over a square that is large enough to completely contain the unit circle. Project each point onto the unit circle. The distribution is not uniform since more points fall on a line from the origin to a vertex of the square than fall on a line from the origin to the midpoint of an edge of the square due to the difference in length. To solve this problem, discard all points outside the unit circle and project the remaining points onto the circle.

In higher dimensions, this method does not work since the fraction of points that fall inside the ball drops to zero and all of the points would be thrown away. The solution is to generate a point each of whose coordinates is an independent Gaussian variable. Generate $x_1,x_2,\ldots,x_d$, using a zero mean, unit variance Gaussian, namely, $\frac{1}{\sqrt{2\pi}}\exp(-x^2/2)$ on the real line.[2] Thus, the probability density of $\mathbf{x}$ is

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}} e^{-\frac{x_1^2+x_2^2+\cdots+x_d^2}{2}}$$

and is spherically symmetric. Normalizing the vector $\mathbf{x} = (x_1,x_2,\ldots,x_d)$ to a unit vector, namely $\frac{\mathbf{x}}{|\mathbf{x}|}$, gives a distribution that is uniform over the surface of the sphere. Note that once the vector is normalized, its coordinates are no longer statistically independent.

To generate a point $\mathbf{y}$ uniformly over the ball (surface and interior), scale the point $\frac{\mathbf{x}}{|\mathbf{x}|}$ generated on the surface by a scalar $\rho \in [0,1]$. What should the distribution of ρ be as a function of r? It is certainly not uniform, even in 2 dimensions. Indeed, the density of ρ at r is proportional to r for $d = 2$. For $d = 3$, it is proportional to r^2. By similar reasoning, the density of ρ at distance r is proportional to r^{d-1} in d dimensions. Solving $\int_{r=0}^{r=1} cr^{d-1}dr = 1$ (the integral of density must equal 1) one should set $c = d$. Another way to see this formally is that the volume of the radius r ball in d dimensions is $r^dV(d)$. The density at radius r is exactly $\frac{d}{dr}(r^dV_d) = dr^{d-1}V_d$. So, pick $\rho(r)$ with density equal to dr^{d-1} for r over [0,1].

We have succeeded in generating a point

$$\mathbf{y} = \rho\frac{\mathbf{x}}{|\mathbf{x}|}$$

uniformly at random from the unit ball by using the convenient spherical Gaussian distribution. In the next sections, we will analyze the spherical Gaussian in more detail.

2.6 Gaussians in High Dimension

A 1-dimensional Gaussian has its mass close to the origin. However, as the dimension is increased something different happens. The d-dimensional spherical Gaussian with zero mean and variance σ^2 in each coordinate has density function

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\,\sigma^d} \exp\left(-\frac{|\mathbf{x}|^2}{2\sigma^2}\right).$$

The value of the density is maximum at the origin, but there is very little volume there. When $\sigma^2 = 1$, integrating the probability density over a unit ball centered at the origin yields almost zero mass since the volume of such a ball is negligible. In fact, one needs

[2]One might naturally ask: "how do you generate a random number from a 1-dimensional Gaussian?" To generate a number from any distribution given its cumulative distribution function P, first select a uniform random number $u \in [0,1]$ and then choose $x = P^{-1}(u)$. For any $a < b$, the probability that x is between a and b is equal to the probability that u is between $P(a)$ and $P(b)$ which equals $P(b) - P(a)$ as desired. For the 2-dimensional Gaussian, one can generate a point in polar coordinates by choosing angle θ uniform in $[0,2\pi]$ and radius $r = \sqrt{-2\ln(u)}$ where u is uniform random in $[0,1]$. This is called the Box-Muller transform.

to increase the radius of the ball to nearly $\sqrt{d}$ before there is a significant volume and hence significant probability mass. If one increases the radius much beyond $\sqrt{d}$, the integral barely increases even though the volume increases since the probability density is dropping off at a much higher rate. The following theorem formally states that nearly all the probability is concentrated in a thin annulus of width $O(1)$ at radius $\sqrt{d}$.

Theorem 2.9 (Gaussian Annulus Theorem) *For a d-dimensional spherical Gaussian with unit variance in each direction, for any $\beta \leq \sqrt{d}$, all but at most $3e^{-c\beta}$ of the probability mass lies within the annulus $\sqrt{d} - \beta \leq |\mathbf{x}| \leq \sqrt{d} + \beta$, where c is a fixed positive constant.*

For a high-level intuition, note that $E(|\mathbf{x}|^2) = \sum_{i=1}^{d} E(x_i^2) = dE(x_1^2) = d$, so the mean

squared distance of a point from the center is d. The Gaussian Annulus Theorem says that the points are tightly concentrated. We call the square root of the mean squared distance, namely d, the radius of the Gaussian.

To prove the Gaussian Annulus Theorem we make use of a tail inequality for sums of independent random variables of bounded moments (Theorem 12.5).

Proof (Gaussian Annulus Theorem): Let $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ be a point selected from a unit variance Gaussian centered at the origin, and let $r = |\mathbf{x}|$. $d - \beta \leq |\mathbf{y}| \leq d + \beta$ is equivalent to $|r - d| \geq \beta$. If $|r - d| \geq \beta$, then multiplying both sides by $r + d$ gives $|r^2 - d| \geq \beta(r + d) \geq \beta d$. So, it suffices to bound the probability that $|r^2 - d| \geq \beta d$.

Rewrite $r^2 - d = (x_1^2 + \ldots + x_d^2) - d = (x_1^2 - 1) + \ldots + (x_d^2 - 1)$ and perform a change of variables: $y_i = x_i^2 - 1$. We want to bound the probability that $|y_1 + \ldots + y_d| \geq \beta d$. Notice that $E(y_i) = E(x_i^2) - 1 = 0$. To apply Theorem 12.5, we need to bound the s^{th} moments of y_i.

For $|x_i| \leq 1$, $|y_i|^s \leq 1$ and for $|x_i| \geq 1$, $|y_i|^s \leq |x_i|^{2s}$. Thus

$$|E(y_i^s)| = E(|y_i|^s) \leq E(1 + x_i^{2s}) = 1 + E(x_i^{2s})$$
$$= 1 + \sqrt{\frac{2}{\pi}} \int_0^\infty x^{2s} e^{-x^2/2} dx$$

Using the substitution $2z = x^2$,

$$|E(y_i^s)| = 1 + \frac{1}{\sqrt{\pi}} \int_0^\infty 2^s z^{s-(1/2)} e^{-z} dz$$
$$\leq 2^s s!.$$

The last inequality is from the Gamma integral.

Since $E(y_i) = 0$, $Var(y_i) = E(y_i^2) \leq 2^2 2 = 8$. Unfortunately, we do not have $|E(y_i^s)| \leq 8s!$ as required in Theorem 12.5. To fix this problem, perform one more change of variables, using $w_i = y_i/2$. Then, $Var(w_i) \leq 2$ and $|E(w_i^s)| \leq 2s!$, and our goal is now to bound the probability that $|w_1 + \ldots + w_d| \geq \frac{\beta\sqrt{d}}{2}$. Applying Theorem 12.5 where $\sigma^2 = 2$ and $n = d$, this occurs with probability less than or equal to $e^{-\frac{\beta}{96}}$ 3 . ∎

In the next sections we will see several uses of the Gaussian Annulus Theorem.

2.7 Random Projection and Johnson-Lindenstrauss Lemma

One of the most frequently used subroutines in tasks involving high dimensional data is nearest neighbor search. In nearest neighbor search we are given a database of n points in $\mathbf{R}^d$ where n and d are usually large. The database can be preprocessed and stored in an efficient data structure. Thereafter, we are presented "query" points in $\mathbf{R}^d$ and are asked to find the nearest or approximately nearest database point to the query point. Since the number of queries is often large, the time to answer each query should be very small, ideally a small function of log$n$ and log$d$, whereas preprocessing time could be larger, namely a polynomial function of n and d. For this and other problems, dimension reduction, where one projects the database points to a k-dimensional space with $k \ll d$ (usually dependent on log$d$) can be very useful so long as the relative distances between points are approximately preserved. We will see using the Gaussian Annulus Theorem that such a projection indeed exists and is simple.

The projection $f : \mathbf{R}^d \rightarrow \mathbf{R}^k$ that we will examine (many related projections are known to work as well) is the following. Pick k Gaussian vectors $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_k}$ in $\mathbf{R}^d$ with unit-variance coordinates. For any vector $\mathbf{v}$, define the projection $f(\mathbf{v})$ by:

$$f(\mathbf{v}) = (\mathbf{u_1} \cdot \mathbf{v}, \mathbf{u_2} \cdot \mathbf{v}, \ldots, \mathbf{u_k} \cdot \mathbf{v}).$$

The projection $f(\mathbf{v})$ is the vector of dot products of $\mathbf{v}$ with the $\mathbf{u_i}$. We will show that with high probability, $|f(\mathbf{v})| \approx \sqrt{k}|\mathbf{v}|$. For any two vectors $\mathbf{v_1}$ and $\mathbf{v_2}$, $f(\mathbf{v_1} - \mathbf{v_2}) = f(\mathbf{v_1}) - f(\mathbf{v_2})$. Thus, to estimate the distance $|\mathbf{v_1} - \mathbf{v_2}|$ between two vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ in $\mathbf{R}^d$, it suffices to compute $|f(\mathbf{v_1}) - f(\mathbf{v_2})| = |f(\mathbf{v_1} - \mathbf{v_2})|$ in the k-dimensional space since the factor of $\sqrt{k}$ is known and one can divide by it. The reason distances increase when we project to a lower dimensional space is that the vectors $\mathbf{u_i}$ are not unit length. Also notice that the vectors $\mathbf{u_i}$ are not orthogonal. If we had required them to be orthogonal, we would have lost statistical independence.

Theorem 2.10 (The Random Projection Theorem) *Let* $\mathbf{v}$ *be a fixed vector in* $\mathbf{R}^d$ *and let f be defined as above. There exists constant* $c > 0$ *such that for* $\varepsilon \in (0,1)$,

$$Prob\left(\left||f(\mathbf{v})| - \sqrt{k}|\mathbf{v}|\right| \geq \varepsilon\sqrt{k}|\mathbf{v}|\right) \leq 3e^{-ck\varepsilon^2},$$

where the probability is taken over the random draws of vectors $\mathbf{u_i}$ *used to construct f.*

Proof: By scaling both sides of the inner inequality by $|\mathbf{v}|$, we may assume that $|\mathbf{v}| = 1$. The sum of independent normally distributed real variables is also normally distributed where the mean and variance are the sums of the individual means and variances. Since $\mathbf{u_i} \cdot \mathbf{v} = \sum_{j=1}^{d} u_{ij} v_j$, the random variable $\mathbf{u_i} \cdot \mathbf{v}$ has Gaussian density with zero mean and unit variance, in particular,

$$Var(\mathbf{u_i} \cdot \mathbf{v}) = Var\left(\sum_{j=1}^{d} v_{ij} v_j\right) = \sum_{j=1}^{d} v_j^2 Var(u_{ij}) = \sum_{j=1}^{d} v_j^2 = 1$$

Since $\mathbf{u_1}\cdot\mathbf{v}, \mathbf{u_2}\cdot\mathbf{v}, \ldots, \mathbf{u_k}\cdot\mathbf{v}$ are independent Gaussian random variables, $f(\mathbf{v})$ is a random vector from a k-dimensional spherical Gaussian with unit variance in each coordinate, and so the theorem follows from the Gaussian Annulus Theorem (Theorem 2.9) with d replaced by k. ∎

The random projection theorem establishes that the probability of the length of the projection of a single vector differing significantly from its expected value is exponentially small in k, the dimension of the target subspace. By a union bound, the probability that any of $O(n^2)$ pairwise differences $|\mathbf{v_i} - \mathbf{v_j}|$ among n vectors $\mathbf{v_1}, \ldots, \mathbf{v_n}$ differs significantly from their expected values is small, provided $k \geq \frac{3}{c\varepsilon^2} \ln n$. Thus, this random projection preserves all relative pairwise distances between points in a set of n points with high probability. This is the content of the Johnson-Lindenstrauss Lemma.

Theorem 2.11 (Johnson-Lindenstrauss Lemma) *For any* $0 < \varepsilon < 1$ *and any integer n, let* $k \geq \frac{3}{c\varepsilon^2} \ln n$ *with c as in Theorem 2.9. For any set of n points in* R^d*, the random projection* $f : R^d \to R^k$ *defined above has the property that for all pairs of points* $\mathbf{v_i}$ *and* $\mathbf{v_j}$*, with probability at least* $1 - 3/2n$*,*

$$(1-\varepsilon)\sqrt{k}\,|\mathbf{v_i} - \mathbf{v_j}| \leq |f(\mathbf{v_i}) - f(\mathbf{v_j})| \leq (1+\varepsilon)\sqrt{k}\,|\mathbf{v_i} - \mathbf{v_j}|.$$

Proof: Applying the Random Projection Theorem (Theorem 2.10), for any fixed $\mathbf{v_i}$ and $\mathbf{v_j}$, the probability that $|f(\mathbf{v_i} - \mathbf{v_j})|$ is outside the range

$$\left[(1-\varepsilon)\sqrt{k}|\mathbf{v_i} - \mathbf{v_j}|, (1+\varepsilon)\sqrt{k}|\mathbf{v_i} - \mathbf{v_j}|\right]$$

is at most $3e^{-ck\varepsilon^2} \leq 3/n^3$ for $k \geq \frac{3\ln n}{c\varepsilon^2}$. Since there are $\binom{n}{2} < n^2/2$ pairs of points, by the union bound, the probability that any pair has a large distortion is less than $\frac{3}{2n}$. ∎

Remark: It is important to note that the conclusion of Theorem 2.11 asserts for all $\mathbf{v_i}$ and $\mathbf{v_j}$, not just for most of them. The weaker assertion for most $\mathbf{v_i}$ and $\mathbf{v_j}$ is typically less useful, since our algorithm for a problem such as nearest-neighbor search might return one of the bad pairs of points. A remarkable aspect of the theorem is that the number of dimensions in the projection is only dependent logarithmically on n. Since k is often much less than d, this is called a dimension reduction technique. In applications, the dominant term is typically the $1/\varepsilon^2$ term.

For the nearest neighbor problem, if the database has n_1 points and n_2 queries are expected during the lifetime of the algorithm, take $n = n_1 + n_2$ and project the database to a random k-dimensional space, for k as in Theorem 2.11. On receiving a query, project the

query to the same subspace and compute nearby database points. The Johnson Lindenstrauss Lemma says that with high probability this will yield the right answer whatever the query. Note that the exponentially small in k probability was useful here in making k only dependent on $\ln n$, rather than n.

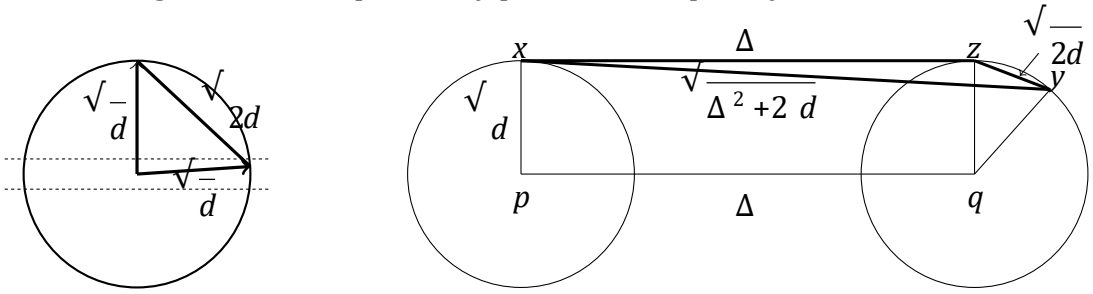
2.8 Separating Gaussians

Mixtures of Gaussians are often used to model heterogeneous data coming from multiple sources. For example, suppose we are recording the heights of individuals age 20-30 in a city. We know that on average, men tend to be taller than women, so a natural model would be a Gaussian mixture model $p(x) = w_1 p_1(x) + w_2 p_2(x)$, where $p_1(x)$ is a Gaussian density representing the typical heights of women, $p_2(x)$ is a Gaussian density representing the typical heights of men, and w_1 and w_2 are the *mixture weights* representing the proportion of women and men in the city. The *parameter estimation problem* for a mixture model is the problem: given access to samples from the overall density p (e.g., heights of people in the city, but without being told whether the person with that height is male or female), reconstruct the parameters for the distribution (e.g., good approximations to the means and variances of p_1 and p_2, as well as the mixture weights).

There are taller women and shorter men, so even if one solved the parameter estimation problem for heights perfectly, given a data point, one couldn't necessarily tell which population it came from. That is, given a height, one couldn't necessarily tell if it came from a man or a woman. In this section, we will look at a problem that is in some ways easier and some ways harder than this problem of heights. It will be harder in that we will be interested in a mixture of two Gaussians in high-dimensions as opposed to the $d = 1$ case of heights. But it will be easier in that we will assume the means are quite well-separated compared to the variances. Specifically, our focus will be on a mixture of two spherical unit-variance Gaussians whose means are separated by a distance $\Omega(d^{1/4})$. We will show that at this level of separation, we can with high probability uniquely determine which Gaussian each data point came from. The algorithm to do so will actually be quite simple. Calculate the distance between all pairs of points. Points whose distance apart is smaller are from the same Gaussian, whereas points whose distance is larger are from different Gaussians. Later, we will see that with more sophisticated algorithms, even a separation of $\Omega(1)$ suffices.

First, consider just one spherical unit-variance Gaussian centered at the origin. From Theorem 2.9, most of its probability mass lies on an annulus of width $O(1)$ at radius $\sqrt{d}$. Also $e^{-|\mathbf{x}|_2/2} = \prod_i e^{-x_{2i}/2}$ and almost all of the mass is within the slab $\{ \mathbf{x} \mid -c \le x_1 \le c \}$, for $c \in O(1)$.

Pick a point **x** from this Gaussian. After picking **x**, rotate the coordinate system to make the first axis align with **x**. Independently pick a second point **y** from



(a) (b)

Figure 2.5: (a) indicates that two randomly chosen points in high dimension are surely almost nearly orthogonal. (b) indicates the distance between a pair of random points from two different unit balls approximating the annuli of two Gaussians.

this Gaussian. The fact that almost all of the probability mass of the Gaussian is within the slab $\{\mathbf{x} \mid -c \leq x_1 \leq c, c \in O(1)\}$ at the equator implies that **y**'s component along **x**'s direction is $O(1)$ with high probability. Thus, **y** is nearly perpendicular to **x**. So, $|\mathbf{x} - \mathbf{y}| \approx \sqrt{|\mathbf{x}|^2 + |\mathbf{y}|^2}$. See Figure 2.5(a). More precisely, since the coordinate system has been rotated so that **x** is at the North Pole, $\mathbf{x} = (\sqrt{d} \pm O(1), 0, \ldots, 0)$. Since **y** is almost on the equator, further rotate the coordinate system so that the component of **y** that is perpendicular to the axis of the North Pole is in the second coordinate. Then $\mathbf{y} = (O(1), \sqrt{d} \pm O(1), 0, \ldots, 0)$. Thus,

$$(\mathbf{x} - \mathbf{y})^2 = d \pm O(\sqrt{d}) + d \pm O(\sqrt{d}) = 2d \pm O(\sqrt{d})$$

and $|\mathbf{x} - \mathbf{y}| = \sqrt{2d} \pm O(1)$ with high probability.

Consider two spherical unit variance Gaussians with centers **p** and **q** separated by a distance Δ. The distance between a randomly chosen point ~~**x from t**~~**x** from the first Gaussian and a randomly chosen point **y** from the second is close to $\Delta^2 + 2d$, since $\mathbf{x}-\mathbf{p}$, $\mathbf{p}-\mathbf{q}$, and $\mathbf{q} - \mathbf{y}$ are nearly mutually perpendicular. Pick **x** and rotate the coordinate system so that **x** is at the North Pole. Let **z** be the North Pole of the ball approximating the second Gaussian. Now pick **y**. Most of the mass of the second Gaussian is within $O(1)$ of the equator perpendicular to $\mathbf{z} - \mathbf{q}$. Also, most of the mass of each Gaussian is within distance $O(1)$ of the respective equators perpendicular to the line $\mathbf{q} - \mathbf{p}$. See Figure 2.5 (b). Thus,

$$|\mathbf{x}-\mathbf{y}|^2 \approx \Delta^2 + |\mathbf{z}-\mathbf{q}|^2 + |\mathbf{q}-\mathbf{y}|^2$$
$$= \Delta^2 + 2d \pm O(\sqrt{d})).$$

To ensure that the distance between two points picked from the same Gaussian are closer to each other than two points picked from different Gaussians requires that the upper limit of the distance between a pair of points from the same Gaussian is at most the lower limit of distance between points from different Gaussians. This requires that $\sqrt{}$ $\sqrt{}$ $\sqrt{}$

$2d+O(1) \leq 2d + \Delta^2 - O(1)$ or $2d+O(\sqrt{d}) \leq 2d+\Delta^2$, which holds when $\Delta \in \omega(d^{1/4})$. Thus, mixtures of spherical Gaussians can be separated in this way, provided their centers are separated by $\omega(d^{1/4})$. If we have n points and want to correctly separate all of them with high probability, we need our individual high-probability statements to hold with probability $1 - 1/poly(n)$,[3] which means our $O(1)$ terms from Theorem 2.9 become $O(\log n)$. So we need to include an extra $O(\log n)$ term in the separation distance.

Algorithm for separating points from two Gaussians: Calculate all pairwise distances between points. The cluster of smallest pairwise distances must come from a single Gaussian. Remove these points. The remaining points come from the second Gaussian.

One can actually separate Gaussians where the centers are much closer. In the next chapter we will use singular value decomposition to separate points from a mixture of two Gaussians when their centers are separated by a distance $O(1)$.

2.9 Fitting a Spherical Gaussian to Data

Given a set of sample points, $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$, in a d-dimensional space, we wish to find the spherical Gaussian that best fits the points. Let f be the unknown Gaussian with mean μ and variance σ^2 in each direction. The probability density for picking these points when sampling according to f is given by

$$c \exp\left(-\frac{(\mathbf{x_1}-\boldsymbol{\mu})^2 + (\mathbf{x_2}-\boldsymbol{\mu})^2 + \cdots + (\mathbf{x_n}-\boldsymbol{\mu})^2}{2\sigma^2}\right)$$

where the normalizing constant c is the reciprocal of $\left[\int e^{-\frac{|\mathbf{x}-\boldsymbol{\mu}|^2}{2\sigma^2}} dx\right]^n$. In integrating from $-\infty$ to ∞, one can shift the origin to μ and thus c is $\left[\int e^{-\frac{|\mathbf{x}|^2}{2\sigma^2}} dx\right]^{-n} = \frac{1}{(2\pi)^{\frac{n}{2}}}$ and is

[3] poly(n) means bounded by a polynomial in n.

independent of μ.

The *Maximum Likelihood Estimator* (MLE) of *f*, given the samples $\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}$, is the *f* that maximizes the above probability density.

Lemma 2.12 *Let* $\{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_n\}$ *be a set of n d-dimensional points. Then* $(\mathbf{x}_1-\mu)^2+(\mathbf{x}_2-\mu)^2+\cdots+(\mathbf{x}_n-\mu)^2$ *is minimized when* μ *is the centroid of the points* $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_n$, *namely* $\mu = \frac{1}{n}(\mathbf{x}_1+\mathbf{x}_2+\cdots+\mathbf{x}_n)$.

Proof: Setting the gradient of $(\mathbf{x}_1-\mu)^2+(\mathbf{x}_2-\mu)^2+\cdots+(\mathbf{x}_n-\mu)^2$ with respect to μ to zero yields

$$-2(\mathbf{x}_1-\mu)-2(\mathbf{x}_2-\mu)-\cdots-2(\mathbf{x}_n-\mu)=0.$$

Solving for μ gives $\mu = \frac{1}{n}(\mathbf{x}_1+\mathbf{x}_2+\cdots+\mathbf{x}_n)$. ■To determine the maximum likelihood estimate of σ^2 for *f*, set μ to the true centroid. Next, show that σ is set to the standard deviation of the sample. Substitute $\nu = \frac{1}{2\sigma^2}$ and $a = (\mathbf{x}_1-\mu)^2+(\mathbf{x}_2-\mu)^2+\cdots+(\mathbf{x}_n-\mu)^2$ into the formula for the probability of picking the points $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_n$. This gives

$$\frac{e^{-a\nu}}{\left[\int_x e^{-x^2\nu}dx\right]^n}.$$

Now, *a* is fixed and *v* is to be determined. Taking logs, the expression to maximize is

$$-a\nu - n\ln\left[\int_x e^{-\nu x^2}dx\right].$$

To find the maximum, differentiate with respect to *v*, set the derivative to zero, and solve for σ. The derivative is

$$-a + n\frac{\int_x |x|^2 e^{-\nu x^2}dx}{\int_x e^{-\nu x^2}dx}.$$

Setting $y = |\sqrt{\nu}\mathbf{x}|$ in the derivative, yields

$$-a + \frac{n}{\nu}\frac{\int_y y^2 e^{-y^2}dy}{\int_y e^{-y^2}dy}.$$

Since the ratio of the two integrals is the expected distance squared of a *d*-dimensional spherical Gaussian of standard deviation $\frac{1}{\sqrt{2}}$ to its center, and this is known to be $\frac{d}{2}$, we get $-a+\frac{nd}{2\nu}$. Substituting σ^2 for $\frac{1}{2\nu}$ gives $-a + nd\sigma^2$. Setting $-a + nd\sigma^2 = 0$ shows that the maximum occurs when $\sigma = \frac{\sqrt{a}}{\sqrt{nd}}$. Note that this quantity is the square root of the average

coordinate distance squared of the samples to their mean, which is the standard deviation of the sample. Thus, we get the following lemma.

Lemma 2.13 *The maximum likelihood spherical Gaussian for a set of samples is the Gaussian with center equal to the sample mean and standard deviation equal to the standard deviation of the sample from the true mean.*

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be a sample of points generated by a Gaussian probability distribution. Then $\mu = \frac{1}{n}(\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_n)$ is an unbiased estimator of the expected value of the distribution. However, if in estimating the variance from the sample set, we use the estimate of the expected value rather than the true expected value, we will not get an unbiased estimate of the variance, since the sample mean is not independent of the sample set. One should use $\tilde{\mu} = \frac{1}{n-1}(\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_n)$ when estimating the variance.
See Section 12.5.10 of the appendix.

2.10 Bibliographic Notes

The word vector model was introduced by Salton [SWY75]. There is vast literature on the Gaussian distribution, its properties, drawing samples according to it, etc. The reader can choose the level and depth according to his/her background. The Master Tail Bounds theorem and the derivation of Chernoff and other inequalities from it are from [Kan09]. The original proof of the Random Projection Theorem by Johnson and Lindenstrauss was complicated. Several authors used Gaussians to simplify the proof. The proof here is due to Dasgupta and Gupta [DG99]. See [Vem04] for details and applications of the theorem. [MU05] and [MR95b] are text books covering much of the material touched upon here.

2.11 Exercises

Exercise 2.1

1. *Let x and y be independent random variables with uniform distribution in* [0,1]. *What is the expected value E(x), E(x²), E(x − y), E(xy), and E(x − y)²?*

2. *Let x and y be independent random variables with uniform distribution in* $[-\frac{1}{2}, \frac{1}{2}]$. *What is the expected value E(x), E(x²), E(x − y), E(xy), and E(x − y)²?*

3. *What is the expected squared distance between two points generated at random inside a unit d-dimensional cube?*

Exercise 2.2 *Randomly generate 30 points inside the cube* $[-\frac{1}{2}, \frac{1}{2}]^{100}$ *and plot distance between points and the angle between the vectors from the origin to the points for all pairs of points.*

Exercise 2.3 *Show that for any a* ≥ 1 *there exist distributions for which Markov's inequality is tight by showing the following:*

1. *For each a* = 2,3, *and 4 give a probability distribution p(x) for a nonnegative random variable x where Prob* $(x \geq a) = \frac{E(x)}{a}$.

2. *For arbitrary a* ≥ 1 *give a probability distribution for a nonnegative random variable x where Prob* $(x \geq a) = \frac{E(x)}{a}$.

Exercise 2.4 *Show that for any c* ≥ 1 *there exist distributions for which Chebyshev's inequality is tight, in other words, Prob*(|*x* − *E*(*x*)| ≥ *c*) = *Var*(*x*)/*c*².

Exercise 2.5 *Let x be a random variable with probability density* $\frac{1}{4}$ *for* 0 ≤ *x* ≤ 4 *and zero elsewhere.*

1. *Use Markov's inequality to bound the probability that x* ≥ 3.

2. *Make use of Prob*(|*x*| ≥ *a*) = *Prob*(*x*² ≥ *a*²) *to get a tighter bound.*

3. *What is the bound using Prob*(|*x*| ≥ *a*) = *Prob*(*x*^*r* ≥ *a*^*r*)?

Exercise 2.6 *Consider the probability distribution* $p(x = 0) = 1 - \frac{1}{a}$ *and* $p(x = a) = \frac{1}{a}$.
Plot the probability that x is greater than or equal to a as a function of a for the bound given by Markov's inequality and by Markov's inequality applied to x² and x⁴.

Exercise 2.7 *Consider the probability density function p*(*x*) = 0 *for x* < 1 *and* $p(x) = c\frac{1}{x^4}$ *for x* ≥ 1.

1. *What should c be to make p a legal probability density function?*

2. *Generate 100 random samples from this distribution. How close is the average of the samples to the expected value of x?*

Exercise 2.8 *Let G be a d-dimensional spherical Gaussian with variance* $\frac{1}{2}$ *in each direction, centered at the origin. Derive the expected squared distance to the origin.*

Exercise 2.9 *Consider drawing a random point* **x** *on the surface of the unit sphere in* R^d*. What is the variance of* x_1 *(the first coordinate of* **x***)? See if you can give an argument without doing any integrals.*

Exercise 2.10 *How large must ε be for 99% of the volume of a* 1000*-dimensional unitradius ball to lie in the shell of ε-thickness at the surface of the ball?*

Exercise 2.11 *Prove that* $1 + x \leq e^x$ *for all real x. For what values of x is the approximation* $1 + x \approx e^x$ *within 0.01?*

Exercise 2.12 *For what value of d does the volume, V* (*d*)*, of a d-dimensional unit ball take on its maximum? Hint: Consider the ratio* $\frac{V(d)}{V(d-1)}$.

Exercise 2.13 *A 3-dimensional cube has vertices, edges, and faces. In a d-dimensional cube, these components are called faces. A vertex is a 0-dimensional face, an edge a 1-dimensional face, etc.*

1. *For* $0 \leq k \leq d$*, how many k-dimensional faces does a d-dimensional cube have?*

2. *What is the total number of faces of all dimensions? The d-dimensional face is the cube itself which you can include in your count.*

3. *What is the surface area of a unit cube in d-dimensions (a unit cube has side-length one in each dimension)?*

4. *What is the surface area of the cube if the length of each side was 2?*

5. *Prove that the volume of a unit cube is close to its surface.*

Exercise 2.14 *Consider the portion of the surface area of a unit radius, 3-dimensional ball with center at the origin that lies within a circular cone whose vertex is at the origin. What is the formula for the incremental unit of area when using polar coordinates to integrate the portion of the surface area of the ball that is lying inside the circular cone? What is the formula for the integral? What is the value of the integral if the angle of the cone is* 36°*? The angle of the cone is measured from the axis of the cone to a ray on the surface of the cone.*

Exercise 2.15 *Consider a unit radius, circular cylinder in 3-dimensions of height one. The top of the cylinder could be an horizontal plane or half of a circular ball. Consider these two possibilities for a unit radius, circular cylinder in 4-dimensions. In 4-dimensions the horizontal plane is 3-dimensional and the half circular ball is 4-dimensional. In each of the*

two cases, what is the surface area of the top face of the cylinder? You can use V (d) for the volume of a unit radius, d-dimension ball and A(d) for the surface area of a unit radius, d-dimensional ball. An infinite length, unit radius, circular cylinder in 4dimensions would be the set $\{(x_1, x_2, x_3, x_4) | x_2^2 + x_3^2 + x_4^2 \leq 1\}$ *where the coordinate* x_1 *is the axis.*

Exercise 2.16 *Given a d-dimensional circular cylinder of radius r and height h*

1. *What is the surface area in terms of V (d) and A(d)?*
2. *What is the volume?*

Exercise 2.17 *How does the volume of a ball of radius two behave as the dimension of the space increases? What if the radius was larger than two but a constant independent of d? What function of d would the radius need to be for a ball of radius r to have approximately constant volume as the dimension increases? Hint: you may want to use Stirling's approximation,* $n! \approx \left(\frac{n}{e}\right)^n$*, for factorial.*

Exercise 2.18 *If* $\lim_{d \to \infty} V(d) = 0$*, the volume of a d-dimensional ball for sufficiently large d must be less than V (3). How can this be if the d-dimensional ball contains the three dimensional ball?*

Exercise 2.19

1. *Write a recurrence relation for V (d) in terms of V (d − 1) by integrating over* x_1*.*
   Hint: At $x_1 = t$*, the (d − 1)-dimensional volume of the slice is the volume of a (d−1)-dimensional sphere of radius* $\sqrt{1 - t^2}$*. Express this in terms of V (d−1) and write down the integral. You need not evaluate the integral.*
2. *Verify the formula for d = 2 and d = 3 by integrating and comparing with V (2) = π and* $V(3) = \frac{4}{3}\pi$

Exercise 2.20 *Consider a unit ball A centered at the origin and a unit ball B whose center is at distance s from the origin. Suppose that a random point x is drawn from the mixture distribution: "with probability 1/2, draw at random from A; with probability 1/2, draw at random from B". Show that a separation* $s \gg 1/\sqrt{d-1}$ *is sufficient so that Prob(x ∈ A ∩ B) = o(1); i.e., for any* $\epsilon > 0$ *there exists c such that if s ≥ c/* $\sqrt{d-1}$*, then Prob* $(x \in A \cap B) < \epsilon$*. In other words, this extent of separation means that nearly all of the mixture distribution is identifiable.*

Exercise 2.21 *Consider the upper hemisphere of a unit-radius ball in d-dimensions. What is the height of the maximum volume cylinder that can be placed entirely inside the*

hemisphere? As you increase the height of the cylinder, you need to reduce the cylinder's radius so that it will lie entirely within the hemisphere.

Exercise 2.22 *What is the volume of the maximum size d-dimensional hypercube that can be placed entirely inside a unit radius d-dimensional ball?*

Exercise 2.23 *Calculate the ratio of area above the plane* $x_1 = \epsilon$ *to the area of the upper hemisphere of a unit radius ball in d-dimensions for* $\epsilon = 0.001, 0.01, 0.02, 0.03, 0.04, 0.05$ *and for d* = 100 *and d* = 1,000*.*

Exercise 2.24 *Almost all of the volume of a ball in high dimensions lies in a narrow slice of the ball at the equator. However, the narrow slice is determined by the point on the surface of the ball that is designated the North Pole. Explain how this can be true if several different locations are selected for the location of the North Pole giving rise to different equators.*

Exercise 2.25 *Explain how the volume of a ball in high dimensions can simultaneously be in a narrow slice at the equator and also be concentrated in a narrow annulus at the surface of the ball.*

Exercise 2.26 *Generate 500 points uniformly at random on the surface of a unit-radius ball in 50 dimensions. Then randomly generate five additional points. For each of the five new points, calculate a narrow band of width* $\frac{2}{\sqrt{50}}$ *at the equator, assuming the point was the North Pole. How many of the 500 points are in each band corresponding to one of the five equators? How many of the points are in all five bands? How wide do the bands need to be for all points to be in all five bands?*

Exercise 2.27 *Place* 100 *points at random on a d-dimensional unit-radius ball. Assume d is large. Pick a random vector and let it define two parallel hyperplanes on opposite sides of the origin that are equal distance from the origin. How close can the hyperplanes be moved and still have at least a .99 probability that all of the* 100 *points land between them?*

Exercise 2.28 *Let* **x** *and* **y** *be d-dimensional zero mean, unit variance Gaussian vectors. Prove that* **x** *and* **y** *are almost orthogonal by considering their dot product.*

Exercise 2.29 *Prove that with high probability, the angle between two random vectors in a high-dimensional space is at least* 45°*. Hint: use Theorem 2.8.*

Exercise 2.30 *Project the volume of a d-dimensional ball of radius d onto a line through the center. For large d, give an intuitive argument that the projected volume should behave like a Gaussian.*

Exercise 2.31

1. *Write a computer program that generates n points uniformly distributed over the surface of a unit-radius d-dimensional ball.*
2. *Generate 200 points on the surface of a sphere in 50 dimensions.*
3. *Create several random lines through the origin and project the points onto each line. Plot the distribution of points on each line.*
4. *What does your result from (3) say about the surface area of the sphere in relation to the lines, i.e., where is the surface area concentrated relative to each line?*

Exercise 2.32 *If one generates points in d-dimensions with each coordinate a unit vari√ ance Gaussian, the points will approximately lie on the surface of a sphere of radius d.*

1. *What is the distribution when the points are projected onto a random line through the origin?*
2. *If one uses a Gaussian with variance four, where in d-space will the points lie?*

Exercise 2.33 *Randomly generate a 100 points on the surface of a sphere in 3-dimensions and in 100-dimensions. Create a histogram of all distances between the pairs of points in both cases.*

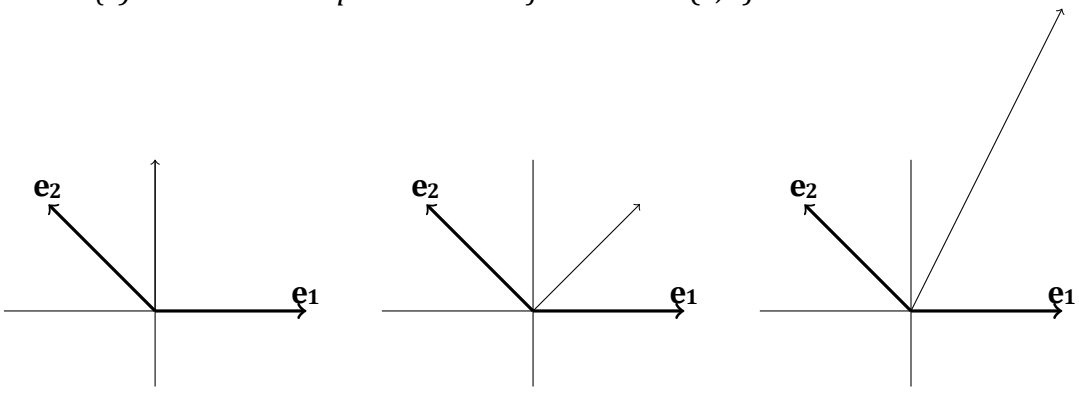
Exercise 2.34 *We have claimed that a randomly generated point on a ball lies near the equator of the ball, independent of the point picked to be the North Pole. Is the same claim true for a randomly generated point on a cube? To test this claim, randomly generate ten* ±1 *valued vectors in 128 dimensions. Think of these ten vectors as ten choices for the North Pole. Then generate some additional* ±1 *valued vectors. To how many of the original vectors is each of the new vectors close to being perpendicular; that is, how many of the equators is each new vector close to?*

Exercise 2.35 *Define the equator of a d-dimensional unit cube to be the hyperplane* $\left\{ \mathbf{x} \middle| \sum_{i=1}^{d} x_i = \frac{d}{2} \right\}$.

1. *Are the vertices of a unit cube concentrated close to the equator?*
2. *Is the volume of a unit cube concentrated close to the equator?*
3. *Is the surface area of a unit cube concentrated close to the equator?*

Exercise 2.36 *Consider a nonorthogonal basis* $\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_d}$. *The* $\mathbf{e_i}$ *are a set of linearly independent unit vectors that span the space.*

1. *Prove that the representation of any vector in this basis is unique.*
2. *Calculate the squared length of* $\mathbf{z} = (\frac{\sqrt{2}}{2}, 1)_e$ *where* $\mathbf{z}$ *is expressed in the basis* $\mathbf{e_1} = (1,0)$ *and* $\mathbf{e^2} = (-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$
3. *If* $\mathbf{y} = \sum_i a_i \mathbf{e_i}$ *and* $\mathbf{z} = \sum_i b_i \mathbf{e_i}$*, with* $0 < a_i < b_i$*, is it necessarily true that the length of* $\mathbf{z}$ *is greater than the length of* $\mathbf{y}$*? Why or why not?*
4. *Consider the basis* $\mathbf{e_1} = (1,0)$ *and* $\mathbf{e^2} = (-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$*.*
   (a) *What is the representation of the vector (0,1) in the basis* $(\mathbf{e_1}, \mathbf{e_2})$*.*
   (b) *What is the representation of the vector* $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$*?*
   (c) *What is the representation of the vector* $(1,2)$*?*



Exercise 2.37 *Generate 20 points uniformly at random on a 900-dimensional sphere of radius 30. Calculate the distance between each pair of points. Then, select a method of projection and project the data onto subspaces of dimension* k*=100, 50, 10, 5, 4, 3, 2, 1 and calculate the difference between* $\sqrt{k}$ *times the original distances and the new pair-wise distances. For each value of k what is the maximum difference as a percent of k.*

Exercise 2.38 *In d-dimensions there are exactly d-unit vectors that are pairwise orthogonal. However, if you wanted a set of vectors that were almost orthogonal you might squeeze in a few more. For example, in 2-dimensions if almost orthogonal meant at least 45 degrees apart, you could fit in three almost orthogonal vectors. Suppose you wanted to find 1000 almost orthogonal vectors in 100 dimensions. Here are two ways you could do it:*

1. *Begin with 1,000 orthonormal 1,000-dimensional vectors, and then project them to a random 100-dimensional space.*
2. *Generate 1000 100-dimensional random Gaussian vectors.*

Implement both ideas and compare them to see which does a better job.

Exercise 2.39 *Suppose there is an object moving at constant velocity along a straight line. You receive the gps coordinates corrupted by Gaussian noise every minute. How do you estimate the current position?*

Exercise 2.40

1. *What is the maximum size rectangle that can be fitted under a unit variance Gaussian?*
2. *What unit area rectangle best approximates a unit variance Gaussian if one measure goodness of fit by the symmetric difference of the Gaussian and the rectangle.*

Exercise 2.41 *Let* $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ *be independent samples of a random variable* $\mathbf{x}$ *with mean* μ *and variance* σ^2*. Let* $\mathbf{m_s} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x_i}$ *be the sample mean. Suppose one estimates the variance using the sample mean rather than the true mean, that is,*

$$\sigma_s^2 = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x_i} - \mathbf{m_s})^2$$

Prove that $E(\sigma_s^2) = \frac{n-1}{n}\sigma^2$ *and thus one should have divided by* $n - 1$ *rather than* n*.*

Hint: First calculate the variance of the sample mean and show that $var(\mathbf{m_s}) = \frac{1}{n} var(\mathbf{x})$*. Then calculate* $E(\sigma_s^2) = E[\frac{1}{n}\sum_{i=1}^{n}(\mathbf{x_i} - \mathbf{m_s})^2]$ *by replacing* $\mathbf{x_i} - \mathbf{m_s}$ *with* $(\mathbf{x_i} - \mathbf{m}) - (\mathbf{m_s} - \mathbf{m})$*.*

Exercise 2.42 *Generate ten values by a Gaussian probability distribution with zero mean and variance one. What is the center determined by averaging the points? What is the variance? In estimating the variance, use both the real center and the estimated center. When using the estimated center to estimate the variance, use both* $n = 10$ *and* $n = 9$*. How do the three estimates compare?*

Exercise 2.43 *Suppose you want to estimate the unknown center of a Gaussian in dspace which has variance one in each direction. Show that* $O(\log d/\varepsilon^2)$ *random samples from the Gaussian are sufficient to get an estimate* $\mathbf{m}_s$ *of the true center* μ*, so that with probability at least 99%,*

$$k\mu - \mathbf{m}_s k_\infty \leq \varepsilon.$$

How many samples are sufficient to ensure that with probability at least 99%

$$k\mu - \mathbf{m}_s k_2 \leq \varepsilon?$$

Exercise 2.44 *Use the probability distribution* $\frac{1}{3\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(x-5)^2}{9}}$ *to generate ten points.*

(a) *From the ten points estimate* μ*. How close is the estimate of* μ *to the true mean of 5?*

(b) *Using the true mean of 5, estimate σ^2 by the formula* $\sigma^2 = \frac{1}{10}\sum_{i=1}^{10}(x_i - 5)^2$. *How close is the estimate of σ^2 to the true variance of 9?*

(c) *Using your estimate m of the mean, estimate σ^2 by the formula* $\sigma^2 = \frac{1}{10}\sum_{i=1}^{10}(x_i - m)^2$. *How close is the estimate of σ^2 to the true variance of 9?*

(d) *Using your estimate m of the mean, estimate σ^2 by the formula* $\sigma^2 = \frac{1}{9}\sum_{i=1}^{10}(x_i - m)^2$. *How close is the estimate of σ^2 to the true variance of 9?*

Exercise 2.45 *Create a list of the five most important things that you learned about high dimensions.*

Exercise 2.46 *Write a short essay whose purpose is to excite a college freshman to learn about high dimensions.*

3 Best-Fit Subspaces and Singular Value Decomposition (SVD)

3.1 Introduction

In this chapter, we examine the *Singular Value Decomposition* (SVD) of a matrix. Consider each row of an $n \times d$ matrix A as a point in d-dimensional space. The singular value decomposition finds the best-fitting k-dimensional subspace for $k = 1,2,3,...,$ for the set of n data points. Here, "best" means minimizing the sum of the squares of the perpendicular distances of the points to the subspace, or equivalently, maximizing the sum of squares of the lengths of the projections of the points onto this subspace.[4] We begin with a special case where the subspace is 1-dimensional, namely a line through the origin. We then show that the best-fitting k-dimensional subspace can be found by k applications of the best fitting line algorithm, where on the i^{th} iteration we find the best fit line perpendicular to the previous $i - 1$ lines. When k reaches the rank of the matrix, from these operations we get an exact decomposition of the matrix called the *singular value decomposition*.

In matrix notation, the singular value decomposition of a matrix A with real entries (we assume all our matrices have real entries) is the factorization of A into the product of three matrices, $A = UDV^T$, where the columns of U and V are orthonormal[5] and the matrix D is diagonal with positive real entries. The columns of V are the unit length vectors defining the best fitting lines described above (the i^{th} column being the unit-length vector in the direction of the i^{th} line). The coordinates of a row of U will be the fractions of the corresponding row of A along the direction of each of the lines.

The SVD is useful in many tasks. Often a data matrix A is close to a low rank matrix and it is useful to find a good low rank approximation to A. For any k, the singular value decomposition of A gives the best rank-k approximation to A in a well-defined sense.

If $\mathbf{u_i}$ and $\mathbf{v_i}$ are columns of U and V respectively, then the matrix equation $A = UDV^T$ can be rewritten as

$$A = \sum_i d_{ii} \mathbf{u_i} \mathbf{v_i}^T.$$

Since $\mathbf{u_i}$ is a $n \times 1$ matrix and $\mathbf{v_i}$ is a $d \times 1$ matrix, $\mathbf{u_i}\mathbf{v_i}^T$ is an $n \times d$ matrix with the same dimensions as A. The i^{th} term in the above sum can be viewed as giving the components of the rows of A along direction $\mathbf{v_i}$. When the terms are summed, they reconstruct A.

[4] This equivalence is due to the Pythagorean Theorem. For each point, its squared length (its distance to the origin squared) is exactly equal to the squared length of its projection onto the subspace plus the squared distance of the point to its projection; therefore, maximizing the sum of the former is equivalent to minimizing the sum of the latter. For further discussion see Section 3.2.

[5] A set of vectors is orthonormal if each is of length one and they are pairwise orthogonal.

This decomposition of A can be viewed as analogous to writing a vector $\mathbf{x}$ in some orthonormal basis $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_d}$. The coordinates of $\mathbf{x} = (\mathbf{x} \cdot \mathbf{v_1}, \mathbf{x} \cdot \mathbf{v_2} \ldots, \mathbf{x} \cdot \mathbf{v_d})$ are the projections of $\mathbf{x}$ onto the $\mathbf{v_i}$'s. For SVD, this basis has the property that for any k, the first k vectors of this basis produce the least possible total sum of squares error for that value of k.

In addition to the singular value decomposition, there is an eigenvalue decomposition. Let A be a square matrix. A vector $\mathbf{v}$ such that $A\mathbf{v} = \lambda\mathbf{v}$ is called an eigenvector and λ the eigenvalue. When A is symmetric, the eigenvectors are orthogonal and A can be expressed as $A = V D V^T$ where the eigenvectors are the columns of V and D is a diagonal matrix with the corresponding eigenvalues on its diagonal. For a symmetric matrix A the singular values and eigenvalues are identical. If the singular values are distinct, then A's singular vectors and eigenvectors are identical. If a singular value has multiplicity d greater than one, the corresponding singular vectors span a subspace of dimension d and any orthogonal basis of the subspace can be used as the eigenvectors or singular vectors.[6]

The singular value decomposition is defined for all matrices, whereas the more familiar eigenvector decomposition requires that the matrix A be square and certain other conditions on the matrix to ensure orthogonality of the eigenvectors. In contrast, the columns of V in the singular value decomposition, called the *right-singular vectors* of A, always form an orthogonal set with no assumptions on A. The columns of U are called the *left-singular vectors* and they also form an orthogonal set (see Section 3.6). A simple consequence of the orthonormality is that for a square and invertible matrix A, the inverse of A is $V D^{-1} U^T$.

Eigenvalues and eignevectors satisfy $A\mathbf{v} = \lambda\mathbf{v}$. We will show that singular values and vectors satisfy a somewhat analogous relationship. Since $A\mathbf{v_i}$ is a $n \times 1$ matrix (vector), the matrix A cannot act on it from the left. But A^T, which is a $d \times n$ matrix, can act on this vector. Indeed, we will show that

$$A\mathbf{v_i} = d_{ii}\mathbf{u_i} \qquad \text{and} \qquad A^T\mathbf{u_i} = d_{ii}\mathbf{v_i}.$$

In words, A acting on $\mathbf{v_i}$ produces a scalar multiple of $\mathbf{u_i}$ and A^T acting on $\mathbf{u_i}$ produces the same scalar multiple of $\mathbf{v_i}$. Note that $A^T A\mathbf{v_i} = d^2_{ii}\mathbf{v_i}$. The i^{th} singular vector of A is the i^{th} eigenvector of the square symmetric matrix $A^T A$.

3.2 Preliminaries

Consider projecting a point $\mathbf{a_i} = (a_{i1}, a_{i2}, \ldots, a_{id})$ onto a line through the origin. Then

[6] When $d = 1$ there are actually two possible singular vectors, one the negative of the other. The subspace spanned is unique.

$a_{i1}^2 + a_{i2}^2 + \cdots + a_{id}^2$= (length of projection)² + (distance of point to line)².

Minimizing $\sum_i$ dist²$_i$ is equivalenttomaximizing $\sum_i$ proj$_i^2$

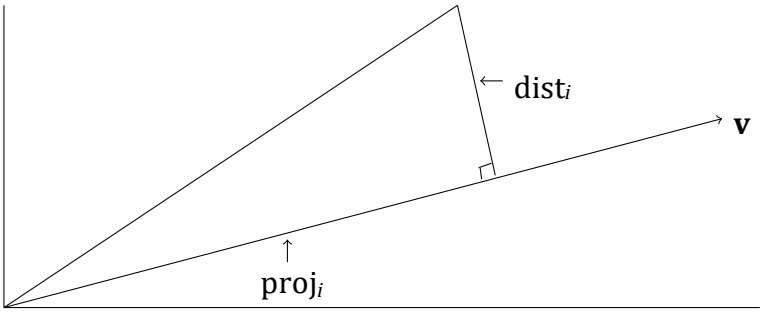


Figure 3.1: The projection of the point $\mathbf{a_i}$ onto the line through the origin in the direction of **v**.

This holds by the Pythagorean Theorem (see Figure 3.1). Thus

$$(\text{distance of point to line})^2 = a_{i1}^2 + a_{i2}^2 + \cdots + a_{id}^2 - (\text{length of projection})^2.$$

Since $\sum_{i=1}^{n}$ ($a^2_{i1} + a^2_{i2} + \cdots + a^2_{id}$) is a constant independent of the line, minimizing the sum of the squares of the distances to the line is equivalent to maximizing the sum of the squares of the lengths of the projections onto the line. Similarly for best-fit subspaces, maximizing the sum of the squared lengths of the projections onto the subspace minimizes the sum of squared distances to the subspace.

Thus we have two interpretations of the best-fit subspace. The first is that it minimizes the sum of squared distances of the data points to it. This first interpretation and its use are akin to the notion of least-squares fit from calculus.[7] The second interpretation of best-fit-subspace is that it maximizes the sum of projections squared of the data points on it. This says that the subspace contains the maximum content of data among all subspaces of the same dimension. The choice of the objective function as the sum of squared distances seems a bit arbitrary and in a way it is. But the square has many nice mathematical properties. The first of these, as we have just seen, is that minimizing the sum of squared distances is equivalent to maximizing the sum of squared projections.

[7] But there is a difference: here we take the perpendicular distance to the line or subspace, whereas, in the calculus notion, given n pairs, $(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)$, we find a line $l = \{(x,y) | y = mx+b\}$ minimizing the vertical squared distances of the points to it, namely, $\sum_{i=1}^{n}(y_i - mx_i - b)^2$.

3.3 Singular Vectors

We now define the *singular vectors* of an $n \times d$ matrix A. Consider the rows of A as n points in a d-dimensional space. Consider the best fit line through the origin. Let $\mathbf{v}$ be a unit vector along this line. The length of the projection of $\mathbf{a_i}$, the i^{th} row of A, onto $\mathbf{v}$ is $|\mathbf{a_i} \cdot \mathbf{v}|$. From this we see that the sum of the squared lengths of the projections is $|A\mathbf{v}|^2$. The best fit line is the one maximizing $|A\mathbf{v}|^2$ and hence minimizing the sum of the squared distances of the points to the line.

With this in mind, define the *first singular vector* $\mathbf{v_1}$ of A as

$$\mathbf{v_1} = \underset{|\mathbf{v}|=1}{\operatorname{argmax}} |A\mathbf{v}|.$$

Technically, there may be a tie for the vector attaining the maximum and so we should not use the article "the"; in fact, $-\mathbf{v_1}$ is always as good as $\mathbf{v_1}$. In this case, we arbitrarily pick one of the vectors achieving the maximum and refer to it as "the first singular vector" avoiding the more cumbersome "one of the vectors achieving the maximum". We adopt this terminology for all uses of argmax.

The value $\sigma_1(A) = |A\mathbf{v_1}|$ is called the *first singular value* of A. Note that $\sigma_1^2 = \sum_{i=1}^{n} (\mathbf{a_i} \cdot \mathbf{v_1})^2$ is the sum of the squared lengths of the projections of the points onto the line determined by $\mathbf{v_1}$.

If the data points were all either on a line or close to a line, intuitively, $\mathbf{v_1}$ should give us the direction of that line. It is possible that data points are not close to one line, but lie close to a 2-dimensional subspace or more generally a low dimensional space. Suppose we have an algorithm for finding $\mathbf{v_1}$ (we will describe one such algorithm later). How do we use this to find the best-fit 2-dimensional plane or more generally the best fit k-dimensional space?

The greedy approach begins by finding $\mathbf{v_1}$ and then finds the best 2-dimensional subspace containing $\mathbf{v_1}$. The sum of squared distances helps. For every 2-dimensional subspace containing $\mathbf{v_1}$, the sum of squared lengths of the projections onto the subspace equals the sum of squared projections onto $\mathbf{v_1}$ plus the sum of squared projections along a vector perpendicular to $\mathbf{v_1}$ in the subspace. Thus, instead of looking for the best 2dimensional subspace containing $\mathbf{v_1}$, look for a unit vector $\mathbf{v_2}$ perpendicular to $\mathbf{v_1}$ that maximizes $|A\mathbf{v}|^2$ among all such unit vectors. Using the same greedy strategy to find the best three and higher dimensional subspaces, defines $\mathbf{v_3}, \mathbf{v_4}, \ldots$ in a similar manner. This is captured in the following definitions. There is no apriori guarantee that the greedy

algorithm gives the best fit. But, in fact, the greedy algorithm does work and yields the best-fit subspaces of every dimension as we will show.

The *second singular vector*, $\mathbf{v_2}$, is defined by the best fit line perpendicular to $\mathbf{v_1}$.

$$\mathbf{v_2} = \underset{\substack{\mathbf{v}\perp\mathbf{v1} \\ |\mathbf{v}|=1}}{\operatorname{argmax}} |A\mathbf{v}|$$

The value $\sigma_2(A) = |A\mathbf{v_2}|$ is called the *second singular value* of *A***.** The *third singular vector* $\mathbf{v}_3$ and the *third singular value* are defined similarly by

$$\mathbf{v_3} = \underset{\mathbf{v}\perp\mathbf{v_1},\mathbf{v_2}\,|\mathbf{v}|=1}{\operatorname{argmax}} |A\mathbf{v}|$$

and $\sigma_3(A) = |A\mathbf{v_3}|$,

and so on. The process stops when we have found singular vectors $\mathbf{v_1},\mathbf{v_2},\ldots,\mathbf{v_r}$, singular values $\sigma_1,\sigma_2,\ldots,\sigma_r$, and $\max |A\mathbf{v}| = 0$.

$$\mathbf{v}\perp\mathbf{v_1},\mathbf{v_2},\ldots,\mathbf{v_r}\ |\mathbf{v}|=1$$

The greedy algorithm found the $\mathbf{v_1}$ that maximized $|A\mathbf{v}|$ and then the best fit 2dimensional subspace containing $\mathbf{v_1}$. Is this necessarily the best-fit 2-dimensional subspace overall? The following theorem establishes that the greedy algorithm finds the best subspaces of every dimension.

Theorem 3.1 (The Greedy Algorithm Works) *Let A be an n×d matrix with singular vectors* $\mathbf{v_1},\mathbf{v_2},\ldots,\mathbf{v_r}$. *For* $1 \le k \le r$, *let* V_k *be the subspace spanned by* $\mathbf{v_1},\mathbf{v_2},\ldots,\mathbf{v_k}$. *For each k,* V_k *is the best-fit k-dimensional subspace for A.*

Proof: The statement is obviously true for $k = 1$. For $k = 2$, let W be a best-fit 2dimensional subspace for A. For any orthonormal basis $(\mathbf{w_1},\mathbf{w_2})$ of W, $|A\mathbf{w_1}|^2 + |A\mathbf{w_2}|^2$ is the sum of squared lengths of the projections of the rows of A onto W. Choose an orthonormal basis $(\mathbf{w}_1,\mathbf{w_2})$ of W so that $\mathbf{w_2}$ is perpendicular to $\mathbf{v_1}$. If $\mathbf{v_1}$ is perpendicular to W, any unit vector in W will do as $\mathbf{w_2}$. If not, choose $\mathbf{w_2}$ to be the unit vector in W perpendicular to the projection of $\mathbf{v_1}$ onto W. This makes $\mathbf{w_2}$ perpendicular to $\mathbf{v_1}$.[8] Since $\mathbf{v_1}$ maximizes $|A\mathbf{v}|^2$, it follows that $|A\mathbf{w_1}|^2 \le |A\mathbf{v_1}|^2$. Since $\mathbf{v_2}$ maximizes $|A\mathbf{v}|^2$ over all $\mathbf{v}$ perpendicular to $\mathbf{v_1}$, $|A\mathbf{w_2}|^2 \le |A\mathbf{v_2}|^2$. Thus

$$|A\mathbf{w_1}|^2 + |A\mathbf{w_2}|^2 \le |A\mathbf{v_1}|^2 + |A\mathbf{v_2}|^2.$$

[8] This can be seen by noting that $\mathbf{v_1}$ is the sum of two vectors that each are individually perpendicular to $\mathbf{w_2}$, namely the projection of $\mathbf{v_1}$ to W and the portion of $\mathbf{v_1}$ orthogonal to W.

Hence, V_2 is at least as good as W and so is a best-fit 2-dimensional subspace.

For general k, proceed by induction. By the induction hypothesis, V_{k-1} is a best-fit k-1 dimensional subspace. Suppose W is a best-fit k-dimensional subspace. Choose an orthonormal basis $\mathbf{w_1},\mathbf{w_2},\ldots,\mathbf{w_k}$ of W so that $\mathbf{w_k}$ is perpendicular to $\mathbf{v_1},\mathbf{v_2},\ldots,\mathbf{v_{k-1}}$.
Then

$|A\mathbf{w_1}|^2 + |A\mathbf{w_2}|^2 + \cdots + |A\mathbf{w_{k-1}}|^2 \leq |A\mathbf{v_1}|^2 + |A\mathbf{v_2}|^2 + \cdots + |A\mathbf{v_{k-1}}|^2$ since V_{k-1} is

an optimal $k-1$ dimensional subspace. Since $\mathbf{w_k}$ is perpendicular to $\mathbf{v_1},\mathbf{v_2},\ldots,\mathbf{v_{k-1}}$, by

the definition of $\mathbf{v_k}$, $|A\mathbf{w_k}|^2 \leq |A\mathbf{v_k}|^2$. Thus

$|A\mathbf{w_1}|^2 + |A\mathbf{w_2}|^2 + \cdots + |A\mathbf{w_{k-1}}|^2 + |A\mathbf{w_k}|^2 \leq |A\mathbf{v_1}|^2 + |A\mathbf{v_2}|^2 + \cdots + |A\mathbf{v_{k-1}}|^2 + |A\mathbf{v_k}|^2$, proving that V_k

is at least as good as W and hence is optimal. ■

Note that the n-dimensional vector $A\mathbf{v_i}$ is a list of lengths (with signs) of the projections of the rows of A onto $\mathbf{v_i}$. Think of $|A\mathbf{v_i}| = \sigma_i(A)$ as the *component* of the matrix A along $\mathbf{v_i}$. For this interpretation to make sense, it should be true that adding up the squares of the components of A along each of the $\mathbf{v_i}$ gives the square of the "whole content of A". This is indeed the case and is the matrix analogy of decomposing a vector into its components along orthogonal directions.

Consider one row, say $\mathbf{a_j}$, of A. Since $\mathbf{v_1},\mathbf{v_2},\ldots,\mathbf{v_r}$ span the space of all rows of A, $\mathbf{a_j} \cdot \mathbf{v} = 0$ for all $\mathbf{v}$ perpendicular to $\mathbf{v_1},\mathbf{v_2},\ldots,\mathbf{v_r}$. Thus, for each row $\mathbf{a_j}$, $\sum_{i=1}^{r} (\mathbf{a_j} \cdot \mathbf{v_i})^2 = |\mathbf{a_j}|^2$. Summing over all rows j,

$$\sum_{j=1}^{n} |\mathbf{a_j}|^2 = \sum_{j=1}^{n} \sum_{i=1}^{r} (\mathbf{a_j} \cdot \mathbf{v_i})^2 = \sum_{i=1}^{r} \sum_{j=1}^{n} (\mathbf{a_j} \cdot \mathbf{v_i})^2 = \sum_{i=1}^{r} |A\mathbf{v_i}|^2 = \sum_{i=1}^{r} \sigma_i^2(A).$$

n n d

But $\sum_{j=1} |\mathbf{a_j}|^2 = \sum\sum_{j=1\ k=1} a_{jk}^2$, the sum of squares of all the entries of A. Thus, the sum of squares of the singular values of A is indeed the square of the "whole content of A", i.e., the sum of squares of all the entries. There is an important norm associated with this quantity, the Frobenius norm of A, denoted $||A||_F$ defined as

$$||A||_F = \sqrt{\sum_{j,k} a_{jk}^2}.$$

Lemma 3.2 *For any matrix A, the sum of squares of the singular values equals the square of the Frobenius norm. That is,* $\sum \sigma_i^2(A) = ||A||_F^2$.

Proof: By the preceding discussion. ∎

The vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_r}$ are called the *right-singular vectors*. The vectors $A\mathbf{v_i}$ form a fundamental set of vectors and we normalize them to length one by

$$\mathbf{u_i} = \frac{1}{\sigma_i(A)} A\mathbf{v_i}.$$

Later we will show that $\mathbf{u}_i$ similarly maximizes $|\mathbf{u}^T A|$ over all $\mathbf{u}$ perpendicular to $\mathbf{u_1}, \ldots, \mathbf{u}_{i-1}$. These $\mathbf{u}_i$ are called the *left-singular vectors*. Clearly, the right-singular vectors are orthogonal by definition. We will show later that the left-singular vectors are also orthogonal.

3.4 Singular Value Decomposition (SVD)

Let A be an $n \times d$ matrix with singular vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_r}$ and corresponding singular values $\sigma_1, \sigma_2, \ldots, \sigma_r$. The left-singular vectors of A are $\mathbf{u_i} = \frac{1}{\sigma_i} A\mathbf{v_i}$ where $\sigma_i \mathbf{u_i}$ is a vector whose coordinates correspond to the projections of the rows of A onto $\mathbf{v_i}$. Each $\sigma_i \mathbf{u_i} \mathbf{v_i^T}$ is a rank one matrix whose rows are the "$\mathbf{v_i}$ components" of the rows of A, i.e., the projections of the rows of A in the $\mathbf{v_i}$ direction. We will prove that A can be decomposed into a sum of rank one matrices as

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T.$$

Geometrically, each point is decomposed in A into its components along each of the r orthogonal directions given by the $\mathbf{v_i}$. We will also prove this algebraically. We begin with a simple lemma that two matrices A and B are identical if $A\mathbf{v} = B\mathbf{v}$ for all $\mathbf{v}$.

Lemma 3.3 *Matrices A and B are identical if and only if for all vectors* $\mathbf{v}$, $A\mathbf{v} = B\mathbf{v}$.

Proof: Clearly, if $A = B$ then $A\mathbf{v} = B\mathbf{v}$ for all $\mathbf{v}$. For the converse, suppose that $A\mathbf{v} = B\mathbf{v}$ for all $\mathbf{v}$. Let $\mathbf{e_i}$ be the vector that is all zeros except for the i^{th} component which has value one. Now $A\mathbf{e_i}$ is the i^{th} column of A and thus $A = B$ if for each i, $A\mathbf{e_i} = B\mathbf{e_i}$. ∎

Theorem 3.4 *Let A be an n × d matrix with right-singular vectors* $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_r}$, *leftsingular vectors* $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_r}$, *and corresponding singular values* $\sigma_1, \sigma_2, \ldots, \sigma_r$. *Then*

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T. \quad r$$

Proof: We first show that multiplying both A and $\sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$ by $\mathbf{v_j}$ results in equality.

$$\sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T \mathbf{v_j} = \sigma_j \mathbf{u_j} = A\mathbf{v_j}$$

Since any vector $\mathbf{v}$ can be expressed as a linear combination of the singular vectors

plus a vector perpendicular to the $\mathbf{v_i}$, $A\mathbf{v} = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T \mathbf{v}$ for all $\mathbf{v}$ and by Lemma 3.3,

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T.$$ ∎

The decomposition $A = \sum_i \sigma_i \mathbf{u_i} \mathbf{v_i}^T$ is called the *singular value decomposition*, *SVD*, of A. We can rewrite this equation in matrix notation as $A = UDV^T$ where $\mathbf{u_i}$ is the i^{th} column of U, $\mathbf{v_i}^T$ is the i^{th} row of V^T, and D is a diagonal matrix with σ_i as the i^{th} entry on its diagonal. For any matrix A, the sequence of singular values is unique and if the singular values are all distinct, then the sequence of singular vectors is unique up to signs. However, when some set of singular values are equal, the corresponding singular vectors span some subspace. Any set of orthonormal vectors spanning this subspace can be used as the singular vectors.

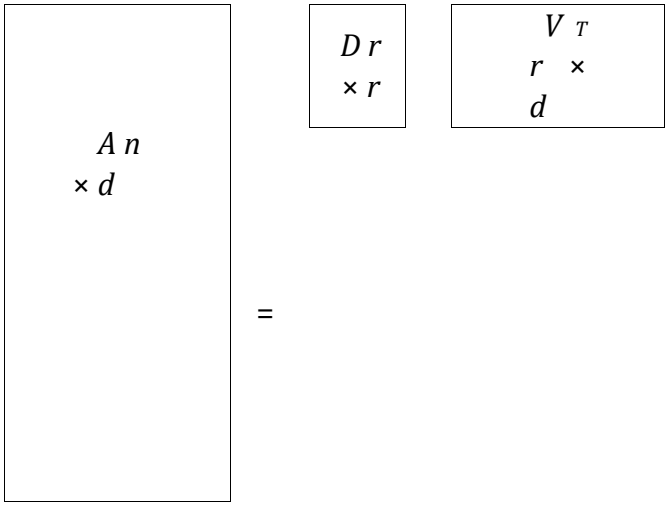


Figure 3.2: The SVD decomposition of an $n \times d$ matrix.

3.5 Best Rank-k Approximations

Let A be an $n \times d$ matrix and think of the rows of A as n points in d-dimensional space. Let

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$$

be the SVD of A. For $k \in \{1,2,\ldots,r\}$, let

$$A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$$

be the sum truncated after k terms. It is clear that A_k has rank k. We show that A_k is the best rank k approximation to A, where error is measured in the Frobenius norm. Geometrically, this says that $\mathbf{v_1},\ldots,\mathbf{v_k}$ define the k-dimensional space minimizing the sum of squared distances of the points to the space. To see why, we need the following lemma.

Lemma 3.5 *The rows of A_k are the projections of the rows of A onto the subspace V_k spanned by the first k singular vectors of A.*

Proof: Let $\mathbf{a}$ be an arbitrary row vector. Since the $\mathbf{v_i}$ are orthonormal, the projection of the vector $\mathbf{a}$ onto V_k is given by $\sum_{i=1}^{k} (\mathbf{a} \cdot \mathbf{v_i}) \mathbf{v_i}^T$. Thus, the matrix whose rows are the projections of the rows of A onto V_k is given by $\sum_{i=1}^{k} A \mathbf{v_i} \mathbf{v_i}^T$. This last expression simplifies to

$$\sum_{i=1}^{k} A \mathbf{v_i} \mathbf{v_i}^T = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i}^T = A_k.$$

■

Theorem 3.6 *For any matrix B of rank at most k*

$$\|A - A_k\|_F \leq \|A - B\|_F$$

Proof: Let B minimize $\|A - B\|_F^2$ among all rank k or less matrices. Let V be the space spanned by the rows of B. The dimension of V is at most k. Since B minimizes $\|A - B\|_F^2$, it must be that each row of B is the projection of the corresponding row of A onto V : Otherwise replace the row of B with the projection of the corresponding row of A onto V . This still keeps the row space of B contained in V and hence the rank of B is still at most k. But it reduces $\|A - B\|_F^2$, contradicting the minimality of $||A - B||_F$.

Since each row of B is the projection of the corresponding row of A, it follows that $\|A - B\|_F^2$ is the sum of squared distances of rows of A to V . Since A_k minimizes the sum of squared distance of rows of A to any k-dimensional subspace, from Theorem 3.1, it follows that $\|A - A_k\|_F \leq \|A - B\|_F$. ■

In addition to the Frobenius norm, there is another matrix norm of interest. Consider an $n \times d$ matrix A and a large number of vectors where for each vector $\mathbf{x}$ we wish to

compute $A\mathbf{x}$. It takes time $O(nd)$ to compute each product $A\mathbf{x}$ but if we approximate A by $A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$ and approximate $A\mathbf{x}$ by $A_k\mathbf{x}$ it requires only k dot products of d-dimensional vectors, followed by a sum of k n-dimensional vectors, and takes time $O(kd + kn)$, which is a win provided $k \ll \min(d, n)$. How is the error measured? Since $\mathbf{x}$ is unknown, the approximation needs to be good for every $\mathbf{x}$. So we take the maximum over all $\mathbf{x}$ of $|(A_k - A)\mathbf{x}|$. Since this would be infinite if $|\mathbf{x}|$ could grow without bound, we restrict the maximum to $|\mathbf{x}| \leq 1$. Formally, we define a new norm of a matrix A by

$$||A||_2 = \max_{|\mathbf{x}| \leq 1} |A\mathbf{x}|.$$

This is called the 2-norm or the spectral norm. Note that it equals $\sigma_1(A)$.

As an application consider a large database of documents that form rows of an $n \times d$ matrix A. There are d terms and each document is a d-dimensional vector with one component for each term, which is the number of occurrences of the term in the document. We are allowed to "preprocess" A. After the preprocessing, we receive queries. Each query $\mathbf{x}$ is an d-dimensional vector which specifies how important each term is to the query. The desired answer is an n-dimensional vector which gives the similarity (dot product) of the query to each document in the database, namely $A\mathbf{x}$, the "matrix-vector" product. Query time is to be much less than preprocessing time, since the idea is that we need to answer many queries for the same database. There are many other applications where one performs many matrix vector products with the same matrix. This technique is applicable to these situations as well.

3.6 Left Singular Vectors

The left singular vectors are also pairwise orthogonal. Intuitively if $\mathbf{u_i}$ and $\mathbf{u_j}$, $i < j$, were not orthogonal, one would suspect that the right singular vector $\mathbf{v_j}$ had a component of $\mathbf{v_i}$ which would contradict that $\mathbf{v_i}$ and $\mathbf{v_j}$ were orthogonal. Let i be the smallest integer such that $\mathbf{u_i}$ is not orthogonal to all other $\mathbf{u_j}$. Then to prove that $\mathbf{u_i}$ and $\mathbf{u_j}$ are orthogonal, we add a small component of $\mathbf{v_j}$ to $\mathbf{v_i}$, normalize the result to be a unit vector

$$\mathbf{v_i}^0 = \frac{\mathbf{v_i} + \epsilon \mathbf{v_j}}{|\mathbf{v_i} + \epsilon \mathbf{v_j}|}$$

and show that $|A\mathbf{v_i'}| > |A\mathbf{v_i}|$, a contradiction.

Theorem 3.7 *The left singular vectors are pairwise orthogonal.*

Proof: Let i be the smallest integer such that $\mathbf{u_i}$ is not orthogonal to some other $\mathbf{u_j}$. Without loss of generality assume that $\mathbf{u}^T_{\mathbf{i}} \mathbf{u_j} = \delta > 0$. If $\mathbf{u_i}^T \mathbf{u_j} < 0$ then just replace $\mathbf{u_i}$ with $-\mathbf{u_i}$. Clearly $j > i$ since i was selected to be the smallest such index. For $\varepsilon > 0$, let

$$= {}^0 \frac{\mathbf{v_i} + \epsilon \mathbf{v_j}\, \mathbf{v_i}}{|\mathbf{v_i} + \epsilon \mathbf{v_j}|}.$$

Notice that $\mathbf{v_i}^0$ is a unit-length vector.

$$A\mathbf{v}_{i0} = \sigma_i \mathbf{u}\sqrt{i1 \overline{++ \varepsilon\sigma\varepsilon_{2j}}}\mathbf{u_j}$$

has length at least as large as its component along $\mathbf{u_i}$ which is

$$\mathbf{u_{Ti}}\left(\frac{\sigma_i \mathbf{u_i} + \varepsilon\sigma_j \mathbf{u_j}}{\sqrt{1+\varepsilon^2}}\right) > (\sigma_i + \varepsilon\sigma_j\delta)\left(1 - \tfrac{\varepsilon^2}{2}\right) > \sigma_i - \tfrac{\varepsilon^2}{2}\sigma_i + \varepsilon\sigma_j\delta - \tfrac{\varepsilon^3}{2}\sigma_j\delta > \sigma_i,$$

for sufficiently small , a contradiction since $\mathbf{v_i}+\varepsilon\mathbf{v_j}$ is orthogonal to $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_{i-1}}$ since $j > i$ and σ_i is defined to be the maximum of $|A\mathbf{v}|$ over such vectors. ∎

Next we prove that A_k is the best rank k, 2-norm approximation to A. We first show that the square of the 2-norm of $A - A_k$ is the square of the $(k+1)^{st}$ singular value of A. This is essentially by definition of A_k; that is, A_k represents the projections of the rows in A onto the space spanned by the top k singular vectors, and so $A - A_k$ is the remaining portion of those rows, whose top singular value will be σ_{k+1}.

Lemma 3.8 $\|A - A_k\|_2^2 = \sigma_{k+1}^2$.

r k

Proof: Let $A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i}\mathbf{v_i}^T$ be the singular value decomposition of A. Then $A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u_i}\mathbf{v_i}^T$

$A - A_k = \sum_{i=k+1}^{r} \sigma_i$ and $\mathbf{u_i}\mathbf{v_i}^T$. Let $\mathbf{v}$ be the top singular vector of $A - A_k$. Express $\mathbf{v}$ as a linear combination of $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_r}$. That is, write $\mathbf{v} = \sum_{j=1}^{r} c_j \mathbf{v_j}$. Then

$$|(A - A_k)\mathbf{v}| = \left|\sum_{i=k+1}^{r} \sigma_i \mathbf{u_i}\mathbf{v_i}^T \sum_{j=1}^{r} c_j \mathbf{v_j}\right| = \left|\sum_{i=k+1}^{r} c_i\sigma_i \mathbf{u_i}\mathbf{v_i}^T \mathbf{v_i}\right|$$

$$= \left|\sum_{i=k+1}^{r} c_i\sigma_i \mathbf{u_i}\right| = \sqrt{\sum_{i=k+1}^{r} c_i^2\sigma_i^2},$$

since the $\mathbf{u_i}$ are orthonormal. The $\mathbf{v}$ maximizing this last quantity, subject to the constraint that $|\mathbf{v}|^2 = \sum_{i=1}^{r} c_i^2 = 1$, occurs when $c_{k+1} = 1$ and the rest of the c_i are zero. Thus, $\|A - A_k\|_2^2 = \sigma_{k+1}^2$ proving the lemma. ∎

Finally, we prove that A_k is the best rank k, 2-norm approximation to A:

Theorem 3.9 *Let A be an $n \times d$ matrix. For any matrix B of rank at most k*

$$\|A - A_k\|_2 \le \|A - B\|_2.$$

Proof: If A is of rank k or less, the theorem is obviously true since $\|A - A_k\|_2 = 0$. Assume that A is of rank greater than k. By Lemma 3.8, $\|A - A_k\|_2^2 = \sigma_{k+1}^2$. The null space of B, the set of vectors $\mathbf{v}$ such that $B\mathbf{v} = 0$, has dimension at least $d - k$. Let $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_{k+1}}$ be the first $k + 1$ singular vectors of A. By a dimension argument, it follows that there exists a $\mathbf{z} \neq 0$ in

$$\text{Null}(B) \cap \text{Span}\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_{k+1}}\}.$$

Scale $\mathbf{z}$ to be of length one.

$$\|A - B\|_2^2 \geq |(A - B)\,\mathbf{z}|^2.$$

Since $B\mathbf{z} = 0$, $\|A - B\|_2^2 \geq |A\mathbf{z}|^2$.

Since $\mathbf{z}$ is in the Span$\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_{k+1}}\}$

$$|A\mathbf{z}|^2 = \left|\sum_{i=1}^{n} \sigma_i \mathbf{u_i} \mathbf{v_i}^T \mathbf{z}\right|^2 = \sum_{i=1}^{n} \sigma_i^2 \left(\mathbf{v_i}^T \mathbf{z}\right)^2 = \sum_{i=1}^{k+1} \sigma_i^2 \left(\mathbf{v_i}^T \mathbf{z}\right)^2 \geq \sigma_{k+1}^2 \sum_{i=1}^{k+1} \left(\mathbf{v_i}^T \mathbf{z}\right)^2 = \sigma_{k+1}^2.$$

It follows that $\|A - B\|_2^2 \geq \sigma_{k+1}^2$ proving the theorem. ∎

For a square symmetric matrix A and eigenvector $\mathbf{v}$, $A\mathbf{v} = \lambda\mathbf{v}$. We now prove the analog for singular values and vectors we discussed in the introduction.

Lemma 3.10 (Analog of eigenvalues and eigenvectors)

$$A\mathbf{v_i} = \sigma_i \mathbf{u_i} \text{ and } A^T \mathbf{u_i} = \sigma_i \mathbf{v_i}.$$

Proof: The first equation follows from the definition of left singular vectors. For the second, note that from the SVD, we get $A^T \mathbf{u_i} = \sum_j \sigma_j \mathbf{v_j} \mathbf{u_j}^T \mathbf{u_i}$, where since the $\mathbf{u_j}$ are orthonormal, all terms in the summation are zero except for $j = i$. ∎

3.7 Power Method for Singular Value Decomposition

Computing the singular value decomposition is an important branch of numerical analysis in which there have been many sophisticated developments over a long period of time. The reader is referred to numerical analysis texts for more details. Here we present an "in-principle" method to establish that the approximate SVD of a matrix A can be computed in polynomial time. The method we present, called the *power method*, is simple and is in fact the conceptual starting point for many algorithms. Let A be a matrix whose SVD is $\sum_i \sigma_i \mathbf{u_i} \mathbf{v_i}^T$. We wish to work with a matrix that is square and symmetric. Let
$B = A^T A$. By direct multiplication, using the orthogonality of the $\mathbf{u_i}$'s that was proved in Theorem 3.7,

$$B = A^T A = \left(\sum_i \sigma_i \mathbf{v_i} \mathbf{u}_i^T\right)\left(\sum_j \sigma_j \mathbf{u_j} \mathbf{v_j}^T\right)$$

$$= \sum_{i,j} \sigma_i \sigma_j \mathbf{v_i} (\mathbf{u}_i^T \cdot \mathbf{u_j}) \mathbf{v_j}^T = \sum_i \sigma_i^2 \mathbf{v_i} \mathbf{v_i}^T.$$

The matrix B is square and symmetric, and has the same left and right-singular vectors. In particular, $B\mathbf{v_j} = (\sum_i \sigma_i^2 \mathbf{v_i} \mathbf{v_i}^T)\mathbf{v_j} = \sigma_j^2 \mathbf{v_j}$, so $\mathbf{v_j}$ is an eigenvector of B with eigenvalue σ_j^2. If A is itself square and symmetric, it will have the same right and left-singular vectors, namely $A = \sum_i \sigma_i \mathbf{v_i} \mathbf{v_i}^T$ and computing B is unnecessary.

Now consider computing B^2.

$$B^2 = \left(\sum_i \sigma_i^2 \mathbf{v_i} \mathbf{v_i}^T\right)\left(\sum_j \sigma_j^2 \mathbf{v_j} \mathbf{v_j}^T\right) = \sum_{ij} \sigma_i^2 \sigma_j^2 \mathbf{v_i} (\mathbf{v_i}^T \mathbf{v_j}) \mathbf{v_j}^T$$

$$r$$

When $i \neq j$, the dot product $\mathbf{v_i}^T \mathbf{v_j}$ is zero by orthogonality.[9] Thus, $B^2 = \sum_{i=1} \sigma_i^4 \mathbf{v_i} \mathbf{v_i}^T$. In computing the k^{th} power of B, all the cross product terms are zero and

$$B^k = \sum_{i=1}^{r} \sigma_i^{2k} \mathbf{v_i} \mathbf{v_i}^T.$$

If $\sigma_1 > \sigma_2$, then the first term in the summation dominates, so $B^k \rightarrow \sigma_1^{2k} \mathbf{v_1} \mathbf{v_1}^T$. This means a close estimate to $\mathbf{v_1}$ can be computed by simply taking the first column of B^k and normalizing it to a unit vector.

3.7.1 A Faster Method

A problem with the above method is that A may be a very large, sparse matrix, say a $10^8 \times 10^8$ matrix with 10^9 nonzero entries. Sparse matrices are often represented by just a list of nonzero entries, say a list of triples of the form (i,j,a_{ij}). Though A is sparse, B need not be and in the worse case may have all 10^{16} entries nonzero[10] and it is then impossible to even

[9] The "outer product" $\mathbf{v_i} \mathbf{v_j}^T$ is a matrix and is not zero even for $i \neq j$.

[10] E.g., suppose each entry in the first row of A is nonzero and the rest of A is zero.

write down B, let alone compute the product B^2. Even if A is moderate in size, computing matrix products is costly in time. Thus, a more efficient method is needed.

Instead of computing B^k, select a random vector $\mathbf{x}$ and compute the product $B^k\mathbf{x}$. The vector $\mathbf{x}$ can be expressed in terms of the singular vectors of B augmented to a full

orthonormal basis as $\mathbf{x} = \sum_{i=1}^{d} c_i\mathbf{v_i}$. Then

$$B^k\mathbf{x} \approx (\sigma_1^{2k}\mathbf{v_1}\mathbf{v_1}^T)\left(\sum_{i=1}^{d} c_i\mathbf{v_i}\right) = \sigma_1^{2k}c_1\mathbf{v_1}.$$

Normalizing the resulting vector yields $\mathbf{v_1}$, the first singular vector of A. The way $B^k\mathbf{x}$ is computed is by a series of matrix vector products, instead of matrix products. $B^k\mathbf{x} = A^T A...A^T A\mathbf{x}$, which can be computed right-to-left. This consists of $2k$ vector times sparse matrix multiplications.

To compute k singular vectors, one selects a random vector $\mathbf{r}$ and finds an orthonormal basis for the space spanned by $\mathbf{r}, A\mathbf{r}, ..., A^{k-1}\mathbf{r}$. Then compute A times each of the basis vectors, and find an orthonormal basis for the space spanned by the resulting vectors. Intuitively, one has applied A to a subspace rather than a single vector. One repeatedly applies A to the subspace, calculating an orthonormal basis after each application to prevent the subspace collapsing to the one dimensional subspace spanned by the first singular vector. The process quickly converges to the first k singular vectors.

An issue occurs if there is no significant gap between the first and second singular values of a matrix. Take for example the case when there is a tie for the first singular vector and $\sigma_1 = \sigma_2$. Then, the above argument fails. We will overcome this hurdle. Theorem 3.11 below states that even with ties, the power method converges to some vector in the span of those singular vectors corresponding to the "nearly highest" singular values. The theorem assumes it is given a vector $\mathbf{x}$ which has a component of magnitude at least δ along the first right singular vector $\mathbf{v_1}$ of A. We will see in Lemma 3.12 that a random vector satisfies this condition with fairly high probability.

Theorem 3.11 *Let A be an $n \times d$ matrix and $\mathbf{x}$ a unit length vector in $\mathbf{R}^d$ with $|\mathbf{x}^T\mathbf{v_1}| \geq \delta$, where $\delta > 0$. Let V be the space spanned by the right singular vectors of A corresponding to singular values greater than $(1-\varepsilon)\sigma_1$. Let $\mathbf{w}$ be the unit vector after $k = \frac{\ln(1/\varepsilon\delta)}{2\varepsilon}$ iterations of the power method, namely,*

$$\mathbf{w} = \frac{(A^TA)^k\mathbf{x}}{\left|(A^TA)^k\mathbf{x}\right|}.$$

Then **w** *has a component of at most ε perpendicular to V.* **Proof:** Let

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$$

be the SVD of A. If the rank of A is less than d, then for convenience complete $\{\mathbf{v_1}, \mathbf{v_2}, \ldots \mathbf{v}_r\}$ into an orthonormal basis $\{\mathbf{v_1}, \mathbf{v_2}, \ldots \mathbf{v}_d\}$ of d-space. Write $\mathbf{x}$ in the basis of the $\mathbf{v_i}$'s as

$$\mathbf{x} = \sum_{i=1}^{d} c_i \mathbf{v_i}.$$

$$(A^T A)^k = \sum_i \sigma_i^{2k} \mathbf{v_i} \mathbf{v_i}^T$$ Since (, it follows that $(A^T A)^k \mathbf{x} = \sum_{i=1}^{d} \sigma_i^{2k} c_i \mathbf{v_i}$. By hypothesis, $|c_1| \geq \delta$.

Suppose that $\sigma_1, \sigma_2, \ldots, \sigma_m$ are the singular values of A that are greater than or equal to $(1 - \varepsilon)\sigma_1$ and that $\sigma_{m+1}, \ldots, \sigma_d$ are the singular values that are less than $(1 - \varepsilon)\sigma_1$. Now

$$|(A^T A)^k \mathbf{x}|^2 = \left| \sum_{i=1}^{d} \sigma_i^{2k} c_i \mathbf{v_i} \right|^2 = \sum_{i=1}^{d} \sigma_i^{4k} c_i^2 \geq \sigma_1^{4k} c_1^2 \geq \sigma_1^{4k} \delta^2.$$

The component of $|(A^T A)^k \mathbf{x}|^2$ perpendicular to the space V is

$$\sum_{i=m+1}^{d} \sigma_i^{4k} c_i^2 \leq (1 - \varepsilon)^2 \sigma_1^{4k} \sum_{i=m+1}^{d} c_i^2 \leq (1 - \varepsilon)^2 \sigma_1^{4k}$$

since $\sum_{i=1}^{d} c_i^2 = |\mathbf{x}|^2 = 1$. Thus, the component of $\mathbf{w}$ perpendicular to V has squared length at most $\frac{(1-\varepsilon)^4 \sigma_1^4}{\sigma_1^{4k} \delta^2}$ and so its length is at most

$$\frac{(1 - \varepsilon)^{2k} \sigma_1^{2k}}{\delta \sigma_1^{2k}} = \frac{(1 - \varepsilon)^{2k}}{\delta} \leq \frac{e^{-2k\varepsilon}}{\delta} = \varepsilon$$

since $k = \frac{\ln(1/\epsilon\delta)}{2\epsilon}$. ∎

Lemma 3.12 *Let* $\mathbf{y} \in \mathbf{R}^n$ *be a random vector with the unit variance spherical Gaussian as its probability density. Normalize y to be a unit length vector by setting* $\mathbf{x} = \mathbf{y}/|\mathbf{y}|$. *Let* $\mathbf{v}$ *be any unit length vector. Then*

$$Prob\left(|\mathbf{x}^T \mathbf{v}| \leq \frac{1}{20\sqrt{d}} \right) \leq \frac{1}{10} + 3e^{-d/96}.$$

Proof: Proving for the unit length vector $\mathbf{x}$ that Probis $\left(|\mathbf{x}^T \mathbf{v}| \leq \frac{1}{20\sqrt{d}} \right) \leq \frac{1}{10} + 3e^{-d/96}$ √

$|\mathbf{y}^T\mathbf{v}| \leq \frac{1}{10}) \leq 1/10$ equivalent to proving for the unnormalized vector$\sqrt{}$ $\mathbf{y}$ that Prob($|\mathbf{y}| \geq 2\ d$) ≤ $3e^{-d/96}$ and Prob(. That Prob($|\mathbf{y}| \geq 2\ d$) is at most $3e^{-d/96}$ follows from Theorem $\sqrt{}$

(2.9) with d substituted for β. The probability that $|\mathbf{y}^T\mathbf{v}| \leq \frac{1}{10}$ is at most 1/10 follows from the fact th~~at~~$\sqrt{}\mathbf{y}^T\mathbf{v}$ is a random, zero mean, unit variance Gaussian with density is at most $1/\sqrt{2\pi} \leq 1/2$ in the interval [−1/10,1/10], so the integral of the Gaussian over the interval is at most 1/10. ∎

3.8 Singular Vectors and Eigenvectors

For a square matrix B, if $B\mathbf{x} = \lambda\mathbf{x}$, then $\mathbf{x}$ is an *eigenvector* of B and λ is the corresponding *eigenvalue*. We saw in Section 3.7, if $B = A^TA$, then the right singular vectors $\mathbf{v_j}$ of A are eigenvectors of B with eigenvalues σ_j^2. The same argument shows that the left singular vectors $\mathbf{u_j}$ of A are eigenvectors of AA^T with eigenvalues σ_j^2.

The matrix $B = A^TA$ has the property that for any vector $\mathbf{x}$, $\mathbf{x}^TB\mathbf{x} \geq 0$. This is because $B = \sum_i \sigma_i^2\mathbf{v_i}\mathbf{v_i}^T$ and for any $\mathbf{x}$, $\mathbf{x}^T\mathbf{v_i}\mathbf{v_i}^T\mathbf{x} = (\mathbf{x}^T\mathbf{v_i})^2 \geq 0$. A matrix B with the property that $\mathbf{x}^TB\mathbf{x} \geq 0$ for all $\mathbf{x}$ is called *positive semi-definite*. Every matrix of the form A^TA is positive semi-definite. In the other direction, any positive semi-definite matrix B can be decomposed into a product A^TA, and so its eigenvalue decomposition can be obtained from the singular value decomposition of A. The interested reader should consult a linear algebra book.

3.9 Applications of Singular Value Decomposition

3.9.1 Centering Data

Singular value decomposition is used in many applications and for some of these applications it is essential to first center the data by subtracting the centroid of the data from each data point.[11] If you are interested in the statistics of the data and how it varies in relationship to its mean, then you would center the data. On the other hand, if you are interested in finding the best low rank approximation to a matrix, then you do not center the data. The issue is whether you are finding the best fitting subspace or the best fitting affine space. In the latter case you first center the data and then find the best fitting subspace. See Figure 3.3.

We first show that the line minimizing the sum of squared distances to a set of points, if not restricted to go through the origin, must pass through the centroid of the points. This implies that if the centroid is subtracted from each data point, such a line will pass through the origin. The best fit line can be generalized to k dimensional "planes". The

[11] The centroid of a set of points is the coordinate-wise average of the points.

operation of subtracting the centroid from all data points is useful in other contexts as well. We give it the name "centering data".

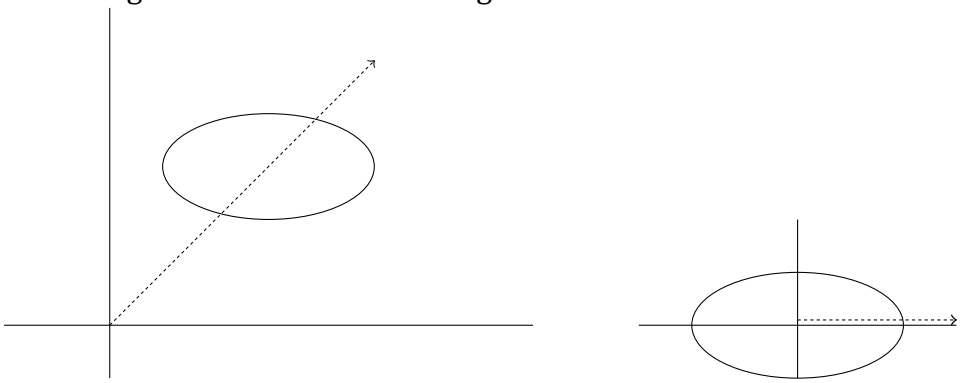


Figure 3.3: If one wants statistical information relative to the mean of the data, one needs to center the data. If one wants the best low rank approximation, one would not center the data.

Lemma 3.13 *The best-fit line (minimizing the sum of perpendicular distances squared) of a set of data points must pass through the centroid of the points.*

Proof: Subtract the centroid from each data point so that the centroid is **0**. After centering the data let ` be the best-fit line and assume for contradiction that ` does not pass through the origin. The line ` can be written as $\{\mathbf{a} + \lambda\mathbf{v} | \lambda \in \mathbf{R}\}$, where **a** is the closest point to **0** on ` and **v** is a unit length vector in the direction of `, which is perpendicular to **a**. For a data point $\mathbf{a_i}$, let $dist(\mathbf{a_i}, \ell)$ denote its perpendicular distance to `. By the Pythagorean theorem, we have $|\mathbf{a_i} - \mathbf{a}|^2 = dist(\mathbf{a_i}, \ell)^2 + (\mathbf{v} \cdot \mathbf{a_i})^2$, or equivalently, $dist(\mathbf{a_i}, \ell)^2 = |\mathbf{a_i} - \mathbf{a}|^2 - (\mathbf{v} \cdot \mathbf{a_i})^2$. Summing over all data points:

$$\sum_{i=1}^{n} dist(\mathbf{a_i}, \ell)^2 = \sum_{i=1}^{n} \left(|\mathbf{a_i} - \mathbf{a}|^2 - (\mathbf{v} \cdot \mathbf{a_i})^2\right) = \sum_{i=1}^{n} \left(|\mathbf{a_i}|^2 + |\mathbf{a}|^2 - 2\mathbf{a_i} \cdot \mathbf{a} - (\mathbf{v} \cdot \mathbf{a_i})^2\right)$$

$$= \sum_{i=1}^{n} |\mathbf{a_i}|^2 + n|\mathbf{a}|^2 - 2\mathbf{a} \cdot \left(\sum_i \mathbf{a_i}\right) - \sum_{i=1}^{n} (\mathbf{v} \cdot \mathbf{a_i})^2 = \sum_i |\mathbf{a_i}|^2 + n|\mathbf{a}|^2 - \sum_i (\mathbf{v} \cdot \mathbf{a_i})^2,$$

where we used the fact that since the centroid is **0**, $\sum_i \mathbf{a_i} = 0$. The above expression is minimized when $\mathbf{a} = \mathbf{0}$, so the line $`' = \{\lambda\mathbf{v} : \lambda \in \mathbf{R}\}$ through the origin is a better fit than `, contradicting ` being the best-fit line. ■

A statement analogous to Lemma 3.13 holds for higher dimensional objects. Define an *affine space* as a subspace translated by a vector. So an affine space is a set of the form

$$\{\mathbf{v_0} + \sum_{i=1}^{k} c_i\mathbf{v_i} | c_1, c_2, \ldots, c_k \in \mathbf{R}\}.$$

Here, $\mathbf{v_0}$ is the translation and $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$ form an orthonormal basis for the subspace.

Lemma 3.14 *The k dimensional affine space which minimizes the sum of squared perpendicular distances to the data points must pass through the centroid of the points.*
Proof: We only give a brief idea of the proof, which is similar to the previous lemma. Instead of $(\mathbf{v} \cdot \mathbf{a_i})^2$, we will now have $\sum_{j=1}^{k} (\mathbf{v_j} \cdot \mathbf{a_i})^2$, where the $\mathbf{v_j}, j = 1,2,\ldots,k$ are an orthonormal basis of the subspace through the origin parallel to the affine space. ■

3.9.2 Principal Component Analysis

The traditional use of SVD is in Principal Component Analysis (PCA). PCA is illustrated by a movie recommendation setting where there are n customers and d movies. Let matrix A with elements a_{ij} represent the amount that customer i likes movie j. One hypothesizes that there are only k underlying basic factors that determine how much a given customer will like a given movie, where k is much smaller than n or d. For example, these could be the amount of comedy, drama, and action, the novelty of the story, etc. Each movie can be described as a k-dimensional vector indicating how much of these basic factors the movie has, and each customer can be described as a k-dimensional vector indicating how important each of these basic factors is to that customer. The dot-product of these two vectors is hypothesized to determine how much that customer will like that movie. In particular, this means that the $n \times d$ matrix A can be expressed as the product of an $n \times k$ matrix U describing the customers and a $k \times d$ matrix V describing the movies. Finding the best rank k approximation A_k by SVD gives such a U and V. One twist is that A may not be exactly equal to UV, in which case $A - UV$ is treated as noise. Another issue is that SVD gives a factorization with negative entries. Nonnegative matrix factorization (NMF) is more appropriate in some contexts where we want to keep entries nonnegative. NMF is discussed in Chapter 9

In the above setting, A was available fully and we wished to find U and V to identify the basic factors. However, in a case such as movie recommendations, each customer may have seen only a small fraction of the movies, so it may be more natural to assume that we are given just a few elements of A and wish to estimate A. If A was an arbitrary matrix of size $n \times d$, this would require $\Omega(nd)$ pieces of information and cannot be done with a few entries. But again hypothesize that A was a small rank matrix with added noise. If now we also assume that the given entries are randomly drawn according to some known distribution, then there is a possibility that SVD can be used to estimate the whole of A. This area is called collaborative filtering and one of its uses is to recommend movies or to target an ad to a customer based on one or two purchases. We do not describe it here.

3.9.3 Clustering a Mixture of Spherical Gaussians

Clustering is the task of partitioning a set of points into k subsets or clusters where each cluster consists of nearby points. Different definitions of the quality of a clustering

lead to different solutions. Clustering is an important area which we will study in detail in Chapter 7. Here we will see how to solve a particular clustering problem using singular value decomposition.

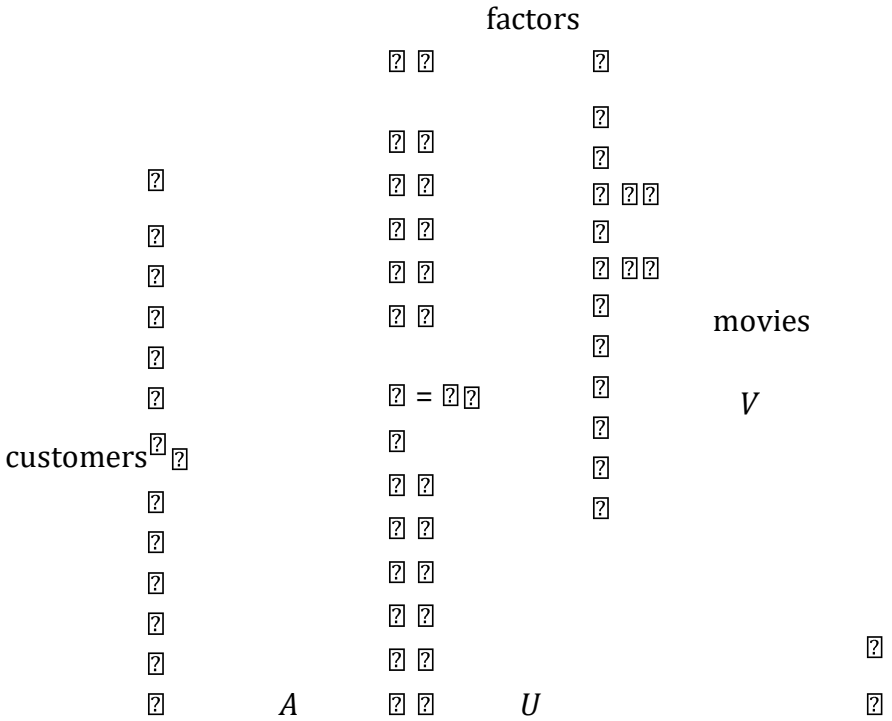


Figure 3.4: Customer-movie data

Mathematical formulations of clustering tend to have the property that finding the highest quality solution to a given set of data is NP-hard. One way around this is to assume stochastic models of input data and devise algorithms to cluster data generated by such models. Mixture models are a very important class of stochastic models. A mixture is a probability density or distribution that is the weighted sum of simple component probability densities. It is of the form

$$f = w_1 p_1 + w_2 p_2 + \cdots + w_k p_k,$$

where $p_1, p_2, \ldots, p_k$ are the basic probability densities and $w_1, w_2, \ldots, w_k$ are positive real numbers called mixture weights that add up to one. Clearly, f is a probability density and integrates to one.

The *model fitting problem* is to fit a mixture of k basic densities to n independent, identically distributed samples, each sample drawn according to the same mixture distribution f. The class of basic densities is known, but various parameters such as their means and the component weights of the mixture are not. Here, we deal with the case where the basic densities are all spherical Gaussians. There are two equivalent ways of thinking of the hidden sample generation process when only the samples are given:

1. Pick each sample according to the density f on $\mathbf{R}^d$.

2. Pick a random i from $\{1,2,...,k\}$ where probability of picking i is w_i. Then, pick a sample according to the density p_i.

One approach to the model-fitting problem is to break it into two subproblems:

1. First, cluster the set of samples into k clusters $C_1,C_2,...,C_k$, where C_i is the set of samples generated according to p_i (see (2) above) by the hidden generation process.

2. Then fit a single Gaussian distribution to each cluster of sample points.

The second problem is relatively easier and indeed we saw the solution in Chapter 2, where we showed that taking the empirical mean (the mean of the sample) and the empirical standard deviation gives us the best-fit Gaussian. The first problem is harder and this is what we discuss here.

If the component Gaussians in the mixture have their centers very close together, then the clustering problem is unresolvable. In the limiting case where a pair of component densities are the same, there is no way to distinguish between them. What condition on the inter-center separation will guarantee unambiguous clustering? First, by looking at 1-dimensional examples, it is clear that this separation should be measured in units of the standard deviation, since the density is a function of the number of standard deviation from the mean. In one dimension, if two Gaussians have inter-center separation at least six times the maximum of their standard deviations, then they hardly overlap. This is summarized in the question: How many standard deviations apart are the means? In one dimension, if the answer is at least six, we can easily tell the Gaussians apart. What is the analog of this in higher dimensions?

We discussed in Chapter 2 distances between two sample points from the same Gaussian as well the distance between two sample points from two different Gaussians. Recall from that discussion that if

- If $\mathbf{x}$ and $\mathbf{y}$ are two independent samples from the same spherical Gaussian with standard deviation[12] σ then

$$|\mathbf{x}-\mathbf{y}|^2 \approx 2\left(\sqrt{d} \pm O(1)\right)^2\sigma^2.$$

- If $\mathbf{x}$ and $\mathbf{y}$ are samples from different spherical Gaussians each of standard deviation σ and means separated by distance Δ, then

[12] Since a spherical Gaussian has the same standard deviation in every direction, we call it the standard deviation of the Gaussian.

$$|\mathbf{x}-\mathbf{y}|^2 \approx 2\left(\sqrt{d} \pm O(1)\right)^2\sigma^2 + \Delta^2.$$

To ensure that points from the same Gaussian are closer to each other than points from different Gaussians, we need

$$2\left(\sqrt{d} - O(1)\right)^2\sigma^2 + \Delta^2 > 2\left(\sqrt{d} + O(1)\right)^2\sigma^2.$$

Expanding the squares, the high order term $2d$ cancels and we need that

$$\Delta > cd^{1/4},$$

for some constant c. While this was not a completely rigorous argument, it can be used to show that a distance based clustering approach (see Chapter 2 for an example) requires an inter-mean separation of at least $cd^{1/4}$ standard deviations to succeed, thus unfortunately not keeping with mnemonic of a constant number of standard deviations separation of the means. Here, indeed, we will show that $\Omega(1)$ standard deviations suffice provided the number k of Gaussians is $O(1)$.

The central idea is the following. Suppose we can find the subspace spanned by the k centers and project the sample points to this subspace. The projection of a spherical Gaussian with standard deviation σ remains a spherical Gaussian with standard deviation σ (Lemma 3.15). In the projection, the inter-center separation remains the same. So in the projection, the Gaussians are distinct provided the inter-center separation in the whole space is at least $ck^{1/4}\sigma$ which is less than $cd^{1/4}\sigma$ for $k \ll d$. Interestingly, we will see that the subspace spanned by the k-centers is essentially the best-fit k-dimensional subspace that can be found by singular value decomposition.

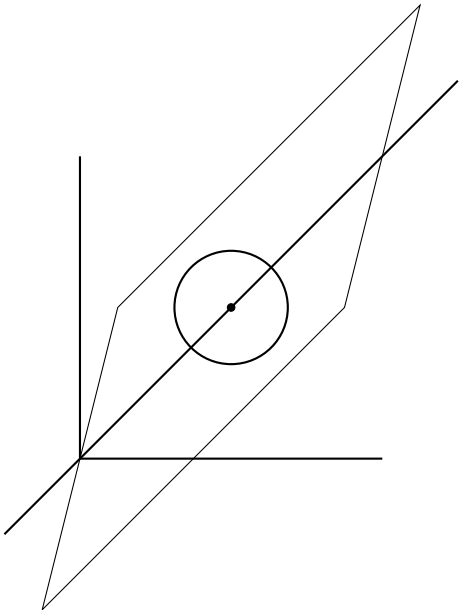
Lemma 3.15 *Suppose p is a d-dimensional spherical Gaussian with center μ and standard deviation σ. The density of p projected onto a k-dimensional subspace V is a spherical Gaussian with the same standard deviation.*

Proof: Rotate the coordinate system so V is spanned by the first k coordinate vectors. The Gaussian remains spherical with standard deviation σ although the coordinates of its center have changed. For a point $\mathbf{x} = (x_1,x_2,\ldots,x_d)$, we will use the notation $\mathbf{x}^0 = (x_1,x_2,\ldots x_k)$ and $\mathbf{x}^{00} = (x_{k+1},x_{k+2},\ldots,x_n)$. The density of the projected Gaussian at the point $(x_1,x_2,\ldots,x_k)$ is

$$ce^{-\frac{|\mathbf{x}'-\boldsymbol{\mu}'|^2}{2\sigma^2}} \int_{\mathbf{x}^{00}} e^{-\frac{|\mathbf{x}''-\boldsymbol{\mu}''|^2}{2\sigma^2}} d\mathbf{x}'' = c'e^{-\frac{|\mathbf{x}'-\boldsymbol{\mu}'|^2}{2\sigma^2}}.$$

This implies the lemma. ∎

We now show that the top k singular vectors produced by the SVD span the space of the k centers. First, we extend the notion of best fit to probability distributions. Then we show that for a single spherical Gaussian whose center is not the origin, the best fit 1-dimensional subspace is the line though the center of the Gaussian and the origin. Next, we show that the best fit k-dimensional subspace for a single Gaussian whose center is not the origin is any k-dimensional subspace containing the line through the Gaussian's center and the origin. Finally, for k spherical Gaussians, the best fit k-dimensional subspace is the subspace containing their centers. Thus, the SVD finds the subspace that contains the centers.



Recall that for a set of points, the best-fit line is the line passing through the origin that maximizes the sum of squared lengths of the projections of the points onto the line.

We extend this definition to probability densities instead of a set of points.

1. The best fit 1-dimension subspace to a spherical Gaussian is the line through its center and the origin.

2. Any k-dimensional subspace containing the line is a best fit k-dimensional subspace for the Gaussian.

3. The best fit k-dimensional subspace for k spherical Gaussians is the subspace containing their centers.

Figure 3.5: Best fit subspace to a spherical Gaussian.

Definition 3.1 *If p is a probability density in d space, the best fit line for p is the line in the* $\mathbf{v_1}$ *direction where* $\mathbf{v_1} = \text{argmax}_{|\mathbf{v}|=1} \; E_{\mathbf{x}\sim p}\left[(\mathbf{v}^T\mathbf{x})^2\right]$.

■

For a spherical Gaussian centered at the origin, it is easy to see that any line passing through the origin is a best fit line. Our next lemma shows that the best fit line for a spherical Gaussian centered at $\mu \neq 0$ is the line passing through μ and the origin.

Lemma 3.16 *Let the probability density p be a spherical Gaussian with center μ 6= 0. The unique best fit 1-dimensional subspace is the line passing through μ and the origin. If μ = 0, then any line through the origin is a best-fit line.*

Proof: For a randomly chosen *x* (according to *p*) and a fixed unit length vector **v,**

$$\begin{aligned}
\underset{\sim p}{E}_{\mathbf{x}}\left[(\mathbf{v}^T\mathbf{x})^2\right] &= \underset{\mathbf{x}\sim p}{E}\left[\left(\mathbf{v}^T(\mathbf{x}-\boldsymbol{\mu})+\mathbf{v}^T\boldsymbol{\mu}\right)^2\right] \\
&= \underset{\mathbf{x}\sim p}{E}\left[\left(\mathbf{v}^T(\mathbf{x}-\boldsymbol{\mu})\right)^2+2\left(\mathbf{v}^T\boldsymbol{\mu}\right)\left(\mathbf{v}^T(\mathbf{x}-\boldsymbol{\mu})\right)+\left(\mathbf{v}^T\boldsymbol{\mu}\right)^2\right] \\
&= \underset{\mathbf{x}\sim p}{E}\left[\left(\mathbf{v}^T(\mathbf{x}-\boldsymbol{\mu})\right)^2\right]+2\left(\mathbf{v}^T\boldsymbol{\mu}\right)E\left[\mathbf{v}^T(\mathbf{x}-\boldsymbol{\mu})\right]+\left(\mathbf{v}^T\boldsymbol{\mu}\right)^2 \\
&= \underset{\mathbf{x}\sim p}{E}\left[\left(\mathbf{v}^T(\mathbf{x}-\boldsymbol{\mu})\right)^2\right]+\left(\mathbf{v}^T\boldsymbol{\mu}\right)^2 \\
&= \sigma^2+\left(\mathbf{v}^T\boldsymbol{\mu}\right)^2
\end{aligned}$$

where the fourth line follows from the fact that $E[\mathbf{v}^T(\mathbf{x}-\mu)] = 0$, and the fifth line follows from the fact that $E[(\mathbf{v}^T(\mathbf{x}-\mu))^2]$ is the variance in the direction **v**. The best fit line **v** maximizes $E_{\mathbf{x}\sim p}[(\mathbf{v}^T\mathbf{x})^2]$ and therefore maximizes $\left(\mathbf{v}^T\boldsymbol{\mu}\right)^2$. This is maximized when **v** is aligned with the center μ. To see uniqueness, just note that if μ 6= 0, then $\mathbf{v}^T\mu$ is strictly less when **v** is not aligned with the center. ■

We now extend Definition 3.1 to *k*-dimensional subspaces.

Definition 3.2 *If p is a probability density in d-space then the best-fit k-dimensional subspace V_k is*

$$V_k = \operatorname{argmax}_{V\ dim(V)=k} E_{\mathbf{x}\sim p}\left(|\mathrm{proj}(\mathbf{x},V)|^2\right),$$

where proj(**x**,*V*) *is the orthogonal projection of* **x** *onto V .* ■

Lemma 3.17 *For a spherical Gaussian with center μ, a k-dimensional subspace is a best fit subspace if and only if it contains μ.*

Proof: If $\mu = \mathbf{0}$, then by symmetry any *k*-dimensional subspace is a best-fit subspace. If μ 6= **0**, then, the best-fit line must pass through μ by Lemma 3.16. Now, as in the greedy algorithm for finding subsequent singular vectors, we would project perpendicular to the first singular vector. But after the projection, the mean of the Gaussian becomes **0** and any vectors will do as subsequent best-fit directions. ■

This leads to the following theorem.

Theorem 3.18 *If p is a mixture of k spherical Gaussians, then the best fit k-dimensional subspace contains the centers. In particular, if the means of the Gaussians are linearly independent, the space spanned by them is the unique best-fit k dimensional subspace.*

Proof: Let p be the mixture $w_1p_1+w_2p_2+\cdots+w_kp_k$. Let V be any subspace of dimension k or less. Then,

$$E_{\mathbf{x}\sim p}\left(|\text{proj}(\mathbf{x},V)|^2\right) = \sum_{i=1}^{k} w_i E_{\mathbf{x}\sim p_i}\left(|\text{proj}(\mathbf{x},V)|^2\right)$$

If V contains the centers of the densities p_i, by Lemma 3.17, each term in the summation is individually maximized, which implies the entire summation is maximized, proving the theorem. ■

For an infinite set of points drawn according to the mixture, the k-dimensional SVD subspace gives exactly the space of the centers. In reality, we have only a large number of samples drawn according to the mixture. However, it is intuitively clear that as the number of samples increases, the set of sample points will approximate the probability density and so the SVD subspace of the sample will be close to the space spanned by the centers. The details of how close it gets as a function of the number of samples are technical and we do not carry this out here.

3.9.4 Ranking Documents and Web Pages

An important task for a document collection is to rank the documents according to their intrinsic relevance to the collection. A good candidate definition of "intrinsic relevance" is a document's projection onto the best-fit direction for that collection, namely the top left-singular vector of the term-document matrix. An intuitive reason for this is that this direction has the maximum sum of squared projections of the collection and so can be thought of as a synthetic term-document vector best representing the document collection.

Ranking in order of the projection of each document's term vector along the best fit direction has a nice interpretation in terms of the power method. For this, we consider a different example, that of the web with hypertext links. The World Wide Web can be represented by a directed graph whose nodes correspond to web pages and directed edges to hypertext links between pages. Some web pages, called *authorities*, are the most prominent sources for information on a given topic. Other pages called *hubs*, are ones that identify the authorities on a topic. Authority pages are pointed to by many hub pages and hub pages point to many authorities. One is led to what seems like a circular definition: a hub is a page that points to many authorities and an authority is a page that is pointed to by many hubs.

One would like to assign hub weights and authority weights to each node of the web. If there are n nodes, the hub weights form an n-dimensional vector $\mathbf{u}$ and the authority weights form an n-dimensional vector $\mathbf{v}$. Suppose A is the adjacency matrix representing the directed graph. Here a_{ij} is 1 if there is a hypertext link from page i to page j and 0 otherwise. Given hub vector $\mathbf{u}$, the authority vector $\mathbf{v}$ could be computed by the formula

$$v_j \propto \sum_{i=1}^{d} u_i a_{ij}$$

since the right hand side is the sum of the hub weights of all the nodes that point to node j. In matrix terms, $\mathbf{v} = A^T\mathbf{u}/|A^T\mathbf{u}|$.

Similarly, given an authority vector $\mathbf{v}$, the hub vector $\mathbf{u}$ could be computed by $\mathbf{u} = A\mathbf{v}/|A\mathbf{v}|$. Of course, at the start, we have neither vector. But the above discussion suggests a power iteration. Start with any $\mathbf{v}$. Set $\mathbf{u} = A\mathbf{v}$, then set $\mathbf{v} = A^T\mathbf{u}$, then renormalize and repeat the process. We know from the power method that this converges to the left and right-singular vectors. So after sufficiently many iterations, we may use the left vector $\mathbf{u}$ as the hub weights vector and project each column of A onto this direction and rank columns (authorities) in order of this projection. But the projections just form the vector $A^T\mathbf{u}$ which equals a multiple of $\mathbf{v}$. So we can just rank by order of the v_j. This is the basis of an algorithm called the HITS algorithm, which was one of the early proposals for ranking web pages.

A different ranking called *pagerank* is widely used. It is based on a random walk on the graph described above. We will study random walks in detail in Chapter 4.

3.9.5 An Application of SVD to a Discrete Optimization Problem

In clustering a mixture of Gaussians, SVD was used as a dimension reduction technique. It found a k-dimensional subspace (the space of centers) of a d-dimensional space and made the Gaussian clustering problem easier by projecting the data to the subspace. Here, instead of fitting a model to data, we consider an optimization problem where applying dimension reduction makes the problem easier. The use of SVD to solve discrete optimization problems is a relatively new subject with many applications. We start with an important NP-hard problem, the maximum cut problem for a directed graph $G(V,E)$.

The maximum cut problem is to partition the nodes of an n-node directed graph into two subsets S and $\bar{S}$ so that the number of edges from S to $\bar{S}$ is maximized. Let A be the adjacency matrix of the graph. With each vertex i, associate an indicator variable x_i. The variable x_i will be set to 1 for $i \in S$ and 0 for $i \in \bar{S}$. The vector $\mathbf{x} = (x_1,x_2,\ldots,x_n)$ is unknown and we are trying to find it or equivalently the cut, so as to maximize the number of edges across the cut. The number of edges across the cut is precisely

X

$$\sum_{i,j} x_i(1 - x_j)a_{ij}.$$

Thus, the maximum cut problem can be posed as the optimization problem

$$\text{Maximize} \sum_{i,j} x_i(1 - x_j)a_{ij} \qquad \text{subject to } x_i \in \{0,1\}.$$

In matrix notation,

$$\sum_{i,j} x_i(1 - x_j)a_{ij} = \mathbf{x}^T A(\mathbf{1} - \mathbf{x}),$$

where $\mathbf{1}$ denotes the vector of all 1's . So, the problem can be restated as

$$\text{Maximize } \mathbf{x}^T A(\mathbf{1} - \mathbf{x}) \qquad \text{subject to } x_i \in \{0,1\}. \tag{3.1}$$

This problem is NP-hard. However we will see that for dense graphs, that is, graphs with $\Omega(n^2)$ edges and therefore whose optimal solution has size $\Omega(n^2)$,[13] we can use the SVD to find a near optimal solution in polynomial time. To do so we will begin by computing the SVD of A and replacing A by $A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$ in (3.1) to get

$$\text{Maximize } \mathbf{x}^T A_k(\mathbf{1} - \mathbf{x}) \qquad \text{subject to } x_i \in \{0,1\}. \tag{3.2}$$

Note that the matrix A_k is no longer a 0-1 adjacency matrix.

We will show that:

1. For each 0-1 vector $\mathbf{x}$, $\mathbf{x}^T A_k(\mathbf{1} - \mathbf{x})$ and $\mathbf{x}^T A(\mathbf{1} - \mathbf{x})$ differ by at most $\frac{n^2}{\sqrt{k+1}}$. Thus, the maxima in (3.1) and (3.2) differ by at most this amount.

2. A near optimal $\mathbf{x}$ for (3.2) can be found in time $n^{O(k)}$ by exploiting the low rank of A_k, which is polynomial time for constant k. By Item 1 this is near optimal for (3.1) where near optimal means with additive error of at most $\frac{n^2}{\sqrt{k+1}}$.

√ First, we prove Item 1. Since $\mathbf{x}$ and $\mathbf{1} - \mathbf{x}$ are 0-1 n-vectors, each has length at most√ n. By the definition of the 2-norm, $|(A - A_k)(\mathbf{1} - \mathbf{x})| \leq n||A - A_k||_2$. Now since $\mathbf{x}^T(A - A_k)(\mathbf{1} - \mathbf{x})$ is the dot product of the vector $\mathbf{x}$ with the vector $(A - A_k)(\mathbf{1} - \mathbf{x})$,

$$|\mathbf{x}^T(A - A_k)(\mathbf{1} - \mathbf{x})| \leq n||A - A_k||_2.$$

[13] Any graph of m edges has a cut of size at least $m/2$. This can be seen by noting that the expected size of the cut for a *random* $\mathbf{x} \in \{0,1\}^n$ is exactly $m/2$.

By Lemma 3.8, $||A - A_k||_2 = \sigma_{k+1}(A)$. The inequalities,

$$(k+1)\sigma_{k2+1} \leq \sigma_{12} + \sigma_{22} + \cdots \sigma_{k2+1} \leq ||A||_{2F} = \sum_{i,j} a_{2ij} \leq n_2$$

imply that $\sigma_{k+1}^2 \leq \frac{n^2}{k+1}$ and hence $||A - A_k||_2 \leq \frac{n}{\sqrt{k+1}}$ proving Item 1.

Next we focus on Item 2. It is instructive to look at the special case when k=1 and A is approximated by the rank one matrix A_1. An even more special case when the left and right-singular vectors **u** and **v** are identical is already NP-hard to solve exactly because it subsumes the problem of whether for a set of n integers, $\{a_1, a_2, \ldots, a_n\}$, there is a partition into two subsets whose sums are equal. However, for that problem, there is an efficient dynamic programming algorithm that finds a near-optimal solution. We will build on that idea for the general rank k problem.

For Item 2, we want to maximize $\sum_{i=1}^{k} \sigma_i (\mathbf{x}^T \mathbf{u_i})(\mathbf{v_i}^T (\mathbf{1} - \mathbf{x}))$ over 0-1 vectors **x**. A piece of notation will be useful. For any $S \subseteq \{1,2,\ldots n\}$, write $\mathbf{u_i}(S)$ for the sum of coordinates of the vector $\mathbf{u_i}$ corresponding to elements in the set S, that is, $\mathbf{u}_i(S) = \sum_{j \in S} u_{ij}$, and similarly for $\mathbf{v_i}$. We will find S to maximize $\sum_{i=1}^{k} \sigma_i \mathbf{u_i}(S)\mathbf{v_i}(\bar{S})$ using dynamic programming.

For a subset S of $\{1,2,\ldots,n\}$, define the $2k$-dimensional vector

$$\mathbf{w}(S) = \left(\mathbf{u_1}(S), \mathbf{v_1}(\bar{S}), \mathbf{u_2}(S), \mathbf{v_2}(\bar{S}), \ldots, \mathbf{u_k}(S), \mathbf{v_k}(\bar{S})\right).$$

If we had the list of all such vectors, we could find $\sum_{i=1}^{k} \sigma_i \mathbf{u_i}(S)\mathbf{v_i}(\bar{S})$ for each of them and take the maximum. There are 2^n subsets S, but several S could have the same $\mathbf{w}(S)$ and in that case it suffices to list just one of them. Round each coordinate of each $\mathbf{u_i}$ to the nearest integer multiple of $\frac{1}{nk^2}$. Call the rounded vector $\tilde{\mathbf{u}}_\mathbf{i}$. Similarly obtain $\tilde{\mathbf{v}}_\mathbf{i}$. Let $\tilde{\mathbf{w}}(S)$ denote the vector $(\tilde{\mathbf{u}}_\mathbf{1}(S), \tilde{\mathbf{v}}_\mathbf{1}(\bar{S}), \tilde{\mathbf{u}}_\mathbf{2}(S), \tilde{\mathbf{v}}_\mathbf{2}(\bar{S}), \ldots, \tilde{\mathbf{u}}_\mathbf{k}(S), \tilde{\mathbf{v}}_\mathbf{k}(\bar{S}))$. We will construct a list of all possible values of the vector $\tilde{\mathbf{w}}(S)$. Again, if several different S's lead to the same vector $\tilde{\mathbf{w}}(S)$, we will keep only one copy on the list. The list will be constructed by dynamic programming. For the recursive step, assume we already have a list of all such vectors for $S \subseteq \{1,2,\ldots,i\}$ and wish to construct the list for $S \subseteq \{1,2,\ldots,i+1\}$. Each $S \subseteq \{1,2,\ldots,i\}$ leads to two possible $S^0 \subseteq \{1,2,\ldots,i+1\}$, namely, S and $S \cup \{i+1\}$.

In the first case, the vector $\tilde{\mathbf{w}}(S^0) = (\tilde{\mathbf{u}}_\mathbf{1}(S), \tilde{\mathbf{v}}_\mathbf{1}(\bar{S}) + \tilde{v}_{1,i+1}, \tilde{\mathbf{u}}_\mathbf{2}(S), \tilde{\mathbf{v}}_\mathbf{2}(\bar{S}) + \tilde{v}_{2,i+1}, \ldots, \ldots)$. In the second case, it is $\tilde{\mathbf{w}}(S^0) = (\tilde{\mathbf{u}}_\mathbf{1}(S) + \tilde{u}_{1,i+1}, \tilde{\mathbf{v}}_\mathbf{1}(\bar{S}), \tilde{\mathbf{u}}_\mathbf{2}(S) + \tilde{u}_{2,i+1}, \tilde{\mathbf{v}}_\mathbf{2}(\bar{S}), \ldots, \ldots)$. We put in these two vectors for each vector in the previous list. Then, crucially, we prune i.e., eliminate duplicates.

Assume that k is constant. Now, we show that the error is at most as $\frac{n^2}{\sqrt{k+1}}$ claimed.

$\sqrt{}$

Since $\mathbf{u_i}$ and $\mathbf{v_i}$ are unit length vectors, $|\mathbf{u_i}(S)|, |\mathbf{v_i}(\bar{S})| \leq n$. Also $|\tilde{\mathbf{u}}_i(S) - \mathbf{u_i}(S)| \leq \frac{n}{nk^2} = \frac{1}{k^2}$ and similarly for $\mathbf{v_i}$. To bound the error, we use an elementary fact: if a and b are reals with $|a|, |b| \leq M$ and we estimate a by a^0 and b by b^0 so that $|a-a^0|, |b-b^0| \leq \delta \leq M$, then $a^0 b^0$ is an estimate of ab in the sense

$|ab - a^0b^0| = |a(b - b^0) + b^0(a - a^0)| \leq |a||b - b^0| + (|b| + |b - b^0|)|a - a^0| \leq 3M\delta$. Using this,

$$\left| \sum_{i=1}^{k} \sigma_i \tilde{\mathbf{u}}_\mathbf{i}(S) \tilde{\mathbf{v}}_\mathbf{i}(\bar{S}) \quad - \quad \sum_{i=1}^{k} \sigma_i \mathbf{u_i}(S) \mathbf{v_i}(S) \right| \leq 3k\sigma_1 \sqrt{n}/k^2 \leq 3n^{3/2}/k \leq n^2/k,$$

and this meets the claimed error bound.

$2\sqrt{n}$Next, we show that the running time is polynomially bounded. First,. Since $\tilde{\mathbf{u}}_\mathbf{i}(S)$ and $\tilde{\mathbf{v}}_\mathbf{i}(S)$ are all integer multiples of $1/(nk^2)$, there are at most$|\tilde{\mathbf{u}}_\mathbf{i}(S)|, |\tilde{\mathbf{v}}2_\mathbf{i}(nS_3)_{/2}| \leq k^2$ possible values of $\tilde{\mathbf{u}}_\mathbf{i}(S)$ and $\tilde{\mathbf{v}}_\mathbf{i}(S)$ from which it follows that the list of $\tilde{\mathbf{w}}(S)$ never gets larger than $(2n^{3/2}k^2)^{2k}$ which for fixed k is polynomially bounded.

We summarize what we have accomplished.

Theorem 3.19 *Given a directed graph $G(V,E)$, a cut of size at least the maximum cut minus $O\left(\frac{n^2}{\sqrt{k}}\right)$ can be computed in time polynomial in n for any fixed k.*

Note that achieving the same accuracy in time polynomial in n and k would give an exact max cut in polynomial time.

3.10 Bibliographic Notes

Singular value decomposition is fundamental to numerical analysis and linear algebra. There are many texts on these subjects and the interested reader may want to study these. A good reference is [GvL96]. The material on clustering a mixture of Gaussians in Section 3.9.3 is from [VW02]. Modeling data with a mixture of Gaussians is a standard tool in statistics. Several well-known heuristics like the expectation-minimization algorithm are used to learn (fit) the mixture model to data. Recently, in theoretical computer science, there has been modest progress on provable polynomial-time algorithms for learning mixtures. Some references are [DS07], [AK05], [AM05], and [MV10]. The application to the discrete optimization problem is from [FK99]. The section on ranking

documents/webpages is from two influential papers, one on hubs and authorities by Jon Kleinberg [Kle99] and the other on pagerank by Page, Brin, Motwani and Winograd [BMPW98].

3.11 Exercises

Exercise 3.1 (Least squares vertical error) *In many experiments one collects the value of a parameter at various instances of time. Let y_i be the value of the parameter y at time x_i. Suppose we wish to construct the best linear approximation to the data in the sense that we wish to minimize the mean square error. Here error is measured vertically rather than perpendicular to the line. Develop formulas for m and b to minimize the mean square error of the points $\{(x_i,y_i)|1 \leq i \leq n\}$ to the line $y = mx + b$.*

Exercise 3.2 *Given five observed variables, height, weight, age, income, and blood pressure of n people, how would one find the best least squares fit affine subspace of the form a_1 (height) + a_2 (weight) + a_3 (age) + a_4 (income) + a_5 (blood pressure) = a_6*

Here $a_1,a_2,\ldots,a_6$ are the unknown parameters. If there is a good best fit 4-dimensional affine subspace, then one can think of the points as lying close to a 4-dimensional sheet rather than points lying in 5-dimensions. Why might it be better to use the perpendicular distance to the affine subspace rather than vertical distance where vertical distance is measured along the coordinate axis corresponding to one of the variables?

Exercise 3.3 *Manually find the best fit lines (not subspaces which must contain the origin) through the points in the sets below. Subtract the center of gravity of the points in the set from each of the points in the set and find the best fit line for the resulting points. Does the best fit line for the original data go through the origin?*

1. *(4,4) (6,2)*
2. *(4,2) (4,4) (6,2) (6,4)*
3. *(3,2.5) (3,5) (5,1) (5,3.5)*

Exercise 3.4 *Manually determine the best fit line through the origin for each of the following sets of points. Is the best fit line unique? Justify your answer for each of the subproblems.*

1. {(0,1),(1,0)}
2. {(0,1),(2,0)}

Exercise 3.5 *Manually find the left and right-singular vectors, the singular values, and the SVD decomposition of the matrices in Figure 3.6.*

Exercise 3.6 *Consider the matrix*

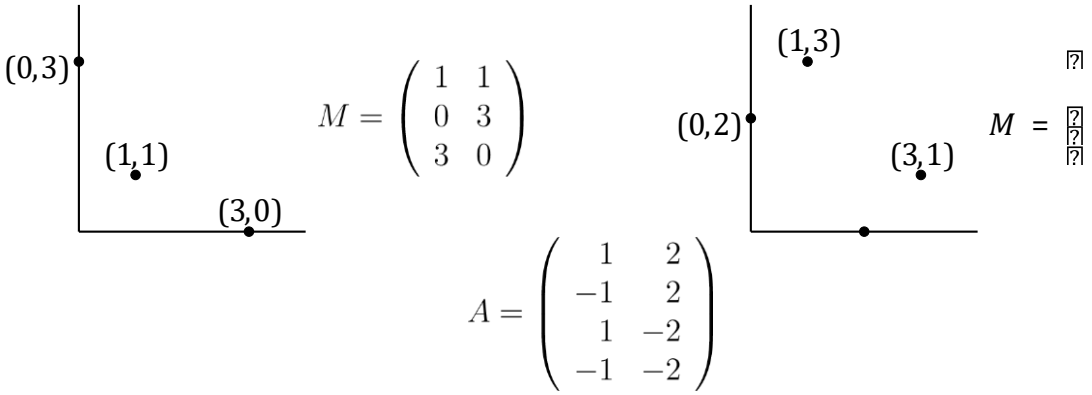


Figure 3.6 a Figure 3.6 b

Figure 3.6: SVD problem

1. *Run the power method starting from* $x = \binom{1}{1}$ *for k = 3 steps. What does this give as an estimate of* v_1*?*

2. *What actually are the* v_i*'s,* σ_i*'s, and* u_i*'s? It may be easiest to do this by computing the eigenvectors of* $B = A^T A$*.*

3. *Suppose matrix A is a database of restaurant ratings: each row is a person, each column is a restaurant, and* a_{ij} *represents how much person i likes restaurant j. What might* v_1 *represent? What about* u_1*? How about the gap* $\sigma_1 - \sigma_2$*?*

Exercise 3.7 *Let A be a square n × n matrix whose rows are orthonormal. Prove that the columns of A are orthonormal.*

Exercise 3.8 *Suppose A is a n×n matrix with block diagonal structure with k equal size blocks where all entries of the* i^{th} *block are* a_i *with* $a_1 > a_2 > \cdots > a_k > 0$*. Show that A has exactly k nonzero singular vectors* $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$ *where* $\mathbf{v_i}$ *has the value* $(\frac{k}{n})^{1/2}$ *in the coordinates corresponding to the* i^{th} *block and 0 elsewhere. In other words, the singular vectors exactly identify the blocks of the diagonal. What happens if* $a_1 = a_2 = \cdots = a_k$*? In the case where the* a_i *are equal, what is the structure of the set of all possible singular vectors?*
Hint: By symmetry, the top singular vector's components must be constant in each block.

Exercise 3.9 *Interpret the first right and left-singular vectors for the document term matrix.*

Exercise 3.10 *Verify that the sum of r-rank one matrices* $\sum_{i=1}^{r} c_i \mathbf{x_i}\mathbf{y_i}^T$ *can be written as* XCY^T*, where the* $\mathbf{x_i}$ *are the columns of X, the* $\mathbf{y_i}$ *are the columns of Y, and C is a diagonal matrix with the constants* c_i *on the diagonal.*

Exercise 3.11 *Let* $\sum_{i=1}^{r} \sigma_i \mathbf{u_i}\mathbf{v_i}^T$ *be the SVD of A. Show that* $\left|\mathbf{u_1}^T A\right| = \sigma_1$ *and that*

$$\left|\mathbf{u_1}^T A\right| = \max_{|\mathbf{u}|=1} \left|\mathbf{u}^{\mathbf{T}} A\right|.$$

Exercise 3.12 *If* $\sigma_1,\sigma_2,...,\sigma_r$ *are the singular values of A and* $\mathbf{v}_1,\mathbf{v}_2,...,\mathbf{v}_r$ *are the corresponding right-singular vectors, show that*

1. $A^T A = \sum_{i=1}^{r} \sigma_i^2 \mathbf{v_i}\mathbf{v_i}^T$
2. $\mathbf{v}_1,\mathbf{v}_2,...\mathbf{v}_r$ *are eigenvectors of* $A^T A$.
3. *Assuming that the eigenvectors of* $A^T A$ *are unique up to multiplicative constants, conclude that the singular vectors of A (which by definition must be unit length) are unique up to sign.*

Exercise 3.13 *Let* $\sum_{i}^{k} \sigma_i u_i v_i^T$ *be the singular value decomposition of a rank r matrix A.*

Let $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$ *be a rank k approximation to A for some k < r. Express the following quantities in terms of the singular values* $\{\sigma_i, 1 \leq i \leq r\}$.

1. $||A_k||^2_F$
2. $||A_k||^2_2$
3. $||A - A_k||^2_F$
4. $||A - A_k||^2_2$

Exercise 3.14 *If A is a symmetric matrix with distinct singular values, show that the left and right singular vectors are the same and that* $A = V D V^T$. **Exercise 3.15** *Let A be a matrix. How would you compute*

$$\mathbf{v_1} = \text{argmax } |A\mathbf{v}|?$$

$|\mathbf{v}|=1$

How would you use or modify your algorithm for finding $\mathbf{v_1}$ *to compute the first few singular vectors of A.*

Exercise 3.16 *Use the power method to compute the singular value decomposition of the matrix*

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

Exercise 3.17 *1. Write a program to implement the power method for computing the first singular vector of a matrix. Apply your program to the matrix*

$$A = \begin{pmatrix} 1 & 2 & 3 & \cdots & 9 & 10 \\ 1 & 3 & 4 & \cdots & 10 & 0 \\ 2 & \ldots & \ldots & \cdots & 0 & \ldots \\ \ldots & 10 & 0 & \cdots & 0 & 0 \\ 9 & & & & & 0 \\ 10 & & & & & \end{pmatrix}.$$

2. Modify the power method to find the first four singular vectors of a matrix A as follows. Randomly select four vectors and find an orthonormal basis for the space spanned by the four vectors. Then multiply each of the basis vectors times A and find a new orthonormal basis for the space spanned by the resulting four vectors. Apply your method to find the first four singular vectors of matrix A from part 1. In Matlab the command orth finds an orthonormal basis for the space spanned by a set of vectors.

Exercise 3.18 *A matrix A is positive semi-definite if for all* $\mathbf{x}$, $\mathbf{x}^T A\mathbf{x} \geq 0$.

1. *Let A be a real valued matrix. Prove that* $B = AA^T$ *is positive semi-definite.*

2. *Let A be the adjacency matrix of a graph. The Laplacian of A is* $L = D - A$ *where D is a diagonal matrix whose diagonal entries are the row sums of A. Prove that L is positive semi definite by showing that* $L = B^T B$ *where B is an m-by-n matrix with a row for each edge in the graph, a column for each vertex, and we define*

$$b_{ei} = \begin{cases} -1 & \text{if } i \text{ is the endpoint of } e \text{ with lesser index if } i \\ 1 & \text{is the endpoint of } e \text{ with greater index if } i \\ 0 & \text{is not an endpoint of } e \end{cases}$$

Exercise 3.19 *Prove that the eigenvalues of a symmetric real valued matrix are real.*

Exercise 3.20 *Suppose A is a square invertible matrix and the SVD of A is* $A = \sum_i \sigma_i u_i v_i^T$.

Prove that the inverse of A is $\sum_i \frac{1}{\sigma_i} v_i u_i^T$.

Exercise 3.21 *Suppose A is square, but not necessarily invertible and has SVD A =* $\sum_{i=1}^{r} \sigma_i u_i v_i^T$. *Let* $B = \sum_{i=1}^{r} \frac{1}{\sigma_i} v_i u_i^T$. *Show that BA***x** = **x** *for all* **x** *in the span of the right-singular vectors of A. For this reason B is sometimes called the pseudo inverse of A and can play the role of* A^{-1} *in many applications.*

Exercise 3.22

1. *For any matrix A, show that* $\sigma_k \leq \frac{||A||_F}{\sqrt{k}}$.

2. *Prove that there exists a matrix B of rank at most k such that* $||A - B||_2 \leq \frac{||A||_F}{\sqrt{k}}$.

3. *Can the 2-norm on the left hand side in (2) be replaced by Frobenius norm?*

Exercise 3.23 *Suppose an n × d matrix A is given and you are allowed to preprocess A. Then you are given a number of d-dimensional vectors* $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}$ *and for each of these vectors you must find the vector* $A\mathbf{x_j}$ *approximately, in the sense that you must find a vector* $\mathbf{y_j}$ *satisfying* $|\mathbf{y_j} - A\mathbf{x_j}| \leq \varepsilon ||A||_F |\mathbf{x_j}|$. *Here ε >0 is a given error bound. Describe an algorithm that accomplishes this in time* $O\left(\frac{d+n}{\varepsilon^2}\right)$ *per* $\mathbf{x_j}$ *not counting the preprocessing time. Hint: use Exercise 3.22.*

Exercise 3.24 *Find the values of* c_i *to maximize* $\sum_{i=1}^{r} c_i^2 \sigma_i^2$ *where* $\sigma_1 \geq \sigma_2 \geq \ldots$ *and* $\sum_{i=1}^{r} c_i^2 = 1$.

Exercise 3.25 (Document-Term Matrices): *Suppose we have an m × n documentterm matrix A where each row corresponds to a document and has been normalized to length one. Define the "similarity" between two such documents by their dot product.*

1. *Consider a "synthetic" document whose sum of squared similarities with all documents in the matrix is as high as possible. What is this synthetic document and how would you find it?*

2. *How does the synthetic document in (1) differ from the center of gravity?*

3. *Building on (1), given a positive integer k, find a set of k synthetic documents such that the sum of squares of the mk similarities between each document in the matrix and each synthetic document is maximized. To avoid the trivial solution of selecting k*

copies of the document in (1), require the k synthetic documents to be orthogonal to each other. Relate these synthetic documents to singular vectors.

4. *Suppose that the documents can be partitioned into k subsets (often called clusters), where documents in the same cluster are similar and documents in different clusters are not very similar. Consider the computational problem of isolating the clusters. This is a hard problem in general. But assume that the terms can also be partitioned into k clusters so that for i* 6= *j, no term in the* i^{th} *cluster occurs in a document in the* j^{th} *cluster. If we knew the clusters and arranged the rows and columns in them to be contiguous, then the matrix would be a block-diagonal matrix. Of course the clusters are not known. By a "block" of the document-term matrix, we mean a submatrix with rows corresponding to the* i^{th}*cluster of documents and columns corresponding to the* i^{th}*cluster of terms . We can also partition any n vector into blocks. Show that any right-singular vector of the matrix must have the property that each of its blocks is a right-singular vector of the corresponding block of the document-term matrix.*

5. *Suppose now that the k singular values are all distinct. Show how to solve the clustering problem.*

Hint: *(4) Use the fact that the right-singular vectors must be eigenvectors of* $A^T A$*. Show that* $A^T A$ *is also block-diagonal and use properties of eigenvectors.*

Exercise 3.26 *Let* $\mathbf{u}$ *be a fixed vector. Show that maximizing* $\mathbf{x}^T \mathbf{u}\mathbf{u}^T (\mathbf{1} - \mathbf{x})$ *subject to* $x_i \in \{0,1\}$ *is equivalent to partitioning the coordinates of* $\mathbf{u}$ *into two subsets where the sum of the elements in both subsets are as equal as possible.*

Exercise 3.27 *Read in a photo and convert to a matrix. Perform a singular value decomposition of the matrix. Reconstruct the photo using only 5%, 10%, 25%, 50% of the singular values.*

1. *Print the reconstructed photo. How good is the quality of the reconstructed photo?*

2. *What percent of the Forbenius norm is captured in each case?*

Hint: *If you use Matlab, the command to read a photo is imread. The types of files that can be read are given by imformats. To print the file use imwrite. Print using jpeg format. To access the file afterwards you may need to add the file extension .jpg. The command imread will read the file in uint8 and you will need to convert to double for the SVD code. Afterwards you will need to convert back to uint8 to write the file. If the photo is a color photo you will get three matrices for the three colors used.*

Exercise 3.28 *1. Create a* 100×100 *matrix of random numbers between 0 and 1 such that each entry is highly correlated with the adjacent entries. Find the SVD of A. What*

fraction of the Frobenius norm of A is captured by the top 10 singular vectors? How many singular vectors are required to capture 95% of the Frobenius norm?

2. *Repeat (1) with a* 100×100 *matrix of statistically independent random numbers between 0 and 1.*

Exercise 3.29 *Show that the running time for the maximum cut algorithm in Section 3.9.5 can be carried out in time* $O(n^3 + poly(n)k^k)$*, where poly is some polynomial.*

Exercise 3.30 *Let* $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ *be n points in d-dimensional space and let X be the n×d matrix whose rows are the n points. Suppose we know only the matrix D of pairwise distances between points and not the coordinates of the points themselves. The set of points* $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ *giving rise to the distance matrix D is not unique since any translation, rotation, or reflection of the coordinate system leaves the distances invariant. Fix the origin of the coordinate system so that the centroid of the set of points is at the origin.*
That is, $\sum_{i=1}^{n} \mathbf{x_i} = 0$.

1. *Show that the elements of* XX^T *are given by*

$$\mathbf{x_i x_j^T} = -\frac{1}{2}\left[d_{ij}^2 - \frac{1}{n}\sum_{k=1}^{n} d_{ik}^2 - \frac{1}{n}\sum_{k=1}^{n} d_{kj}^2 + \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n} d_{kl}^2\right].$$

2. *Describe an algorithm for determining the matrix X whose rows are the* $\mathbf{x_i}$.

Exercise 3.31

1. *Consider the pairwise distance matrix for twenty US cities given below. Use the algorithm of Exercise 3.30 to place the cities on a map of the US. The algorithm is called classical multidimensional scaling, cmdscale, in Matlab. Alternatively use the pairwise distance matrix of 12 Chinese cities to place the cities on a map of China.*

   Note: Any rotation or a mirror image of the map will have the same pairwise distances.

2. *Suppose you had airline distances for 50 cities around the world. Could you use these distances to construct a 3-dimensional world model?*

■

	BOS	BUF	CHI	DAL	DEN	HOU	LAX	MEM	MIA	MSP
Boston	-	400	851	1551	1769	1605	2596	1137	1255	1123
Buffalo	400	-	454	1198	1370	1286	2198	803	1181	731

Chicago	851	454	-	803	920	940	1745	482	1188	355
Dallas	1551	1198	803	-	663	225	1240	420	1111	862
Denver	1769	1370	920	663	-	879	831	879	1726	700
Houston	1605	1286	940	225	879	-	1374	484	968	1056
Los Angeles	2596	2198	1745	1240	831	1374	-	1603	2339	1524
Memphis	1137	803	482	420	879	484	1603	-	872	699
Miami	1255	1181	1188	1111	1726	968	2339	872	-	1511
Minneapolis	1123	731	355	862	700	1056	1524	699	1511	-
New York	188	292	713	1374	1631	1420	2451	957	1092	1018
Omaha	1282	883	432	586	488	794	1315	529	1397	290
Philadelphia	271	279	666	1299	1579	1341	2394	881	1019	985
Phoenix	2300	1906	1453	887	586	1017	357	1263	1982	1280
Pittsburgh	483	178	410	1070	1320	1137	2136	660	1010	743
Saint Louis	1038	662	262	547	796	679	1589	240	1061	466
Salt Lake City	2099	1699	1260	999	371	1200	579	1250	2089	987
San Francisco	2699	2300	1858	1483	949	1645	347	1802	2594	1584
Seattle	2493	2117	1737	1681	1021	1891	959	1867	2734	1395
Washington D.C.	393	292	597	1185	1494	1220	2300	765	923	934

	N	O	P	P	P	S	S	S	S	D
	Y	M	H	H	I	t	L	F	E	C
		A	I	O	T	L	C		A	
Boston	188	1282	271	2300	483	1038	2099	2699	2493	393
Buffalo	292	883	279	1906	178	662	1699	2300	2117	292
Chicago	713	432	666	1453	410	262	1260	1858	1737	597
Dallas	1374	586	1299	887	1070	547	999	1483	1681	1185
Denver	1631	488	1579	586	1320	796	371	949	1021	1494
Houston	1420	794	1341	1017	1137	679	1200	1645	1891	1220
Los Angeles	2451	1315	2394	357	2136	1589	579	347	959	2300
Memphis	957	529	881	1263	660	240	1250	1802	1867	765
Miami	1092	1397	1019	1982	1010	1061	2089	2594	2734	923
Minneapolis	1018	290	985	1280	743	466	987	1584	1395	934
New York	-	1144	83	2145	317	875	1972	2571	2408	230
Omaha	1144	-	1094	1036	836	354	833	1429	1369	1014
Philadelphia	83	1094	-	2083	259	811	1925	2523	2380	123
Phoenix	2145	1036	2083	-	1828	1272	504	653	1114	1973
Pittsburgh	317	836	259	1828	-	559	1668	2264	2138	192
Saint Louis	875	354	811	1272	559	-	1162	1744	1724	712

Salt Lake City	1972	833	1925	504	1668	1162	-	600	701	1848
San Francisco	2571	1429	2523	653	2264	1744	600	-	678	2442
Seattle	2408	1369	2380	1114	2138	1724	701	678	-	2329
Washington D.C.	230	1014	123	1973	192	712	1848	2442	2329	-

City	Bei-jing	Tian-jin	Shang-hai	Chong-qing	Hoh-Urum-hot	Lha-qi	sa	Yin-chuan	Nan-ning	Har-bin	Chang-chun	Shen-yang
Beijing	0	125	1239	3026	480	33003736		1192	2373	1230	979	684
Tianjin	125	0	1150	1954	604	33303740		1316	2389	1207	955	661
Shanghai	1239	1150	0	1945	1717	39294157		2092	1892	2342	2090	1796
Chongqing	3026	1954	1945	0	1847	32022457		1570	993	3156	2905	2610
Hohhot	480	604	1717	1847	0	28253260		716	2657	1710	1458	1164
Urumqi	3300	3330	3929	3202	2825	02668		2111	4279	4531	4279	3985
Lhasa	3736	3740	4157	2457	3260	2668	0	2547	3431	4967	4715	4421
Yinchuan	1192	1316	2092	1570	716	21112547		0	2673	2422	2170	1876
Nanning	2373	2389	1892	993	2657	42793431		2673	0	3592	3340	3046
Harbin	1230	1207	2342	3156	1710	45314967		2422	3592	0	256	546
Changchun	979	955	2090	2905	1458	42794715		2170	3340	256	0	294
Shenyang	684	661	1796	2610	1164	39854421		1876	3046	546	294	0

Exercise 3.32 *One's data in a high dimensional space may lie on a lower dimensional sheath. To test for this one might for each data point find the set of closest data points and calculate the vector distance from the data point to each of the close points. If the set of these distance vectors is a lower dimensional space than the number of distance points, then it is likely that the data is on a low dimensional sheath. To test the dimension of the space of the distance vectors one might use the singular value decomposition to find the singular values. The dimension of the space is the number of large singular values. The low singular values correspond to noise or slight curvature of the sheath. To test this concept generate a data set of points that lie on a one dimensional curve in three space. For each point find maybe ten nearest points, form the matrix of distance, and do a singular value decomposition on the matrix. Report what happens.*

Using code such as the following to create the data.

```
function [ data, distance ] = create_sheath( n )
%creates n data points on a one dimensional sheath in three dimensional
%space % if nargin==0
n=100; end
data=zeros(3,n); for i=1:n
x=sin((pi/100)*i); y=sqrt(1-
x^2); z=0.003*i;
data(:,i)=[x;y;z];
end
```

```
%subtract adjacent vertices distance=zeros(3,10); for
i=1:5 distance(:,i)=data(:,i)-data(:,6);
distance(:,i+5)=data(:,i+6)-data(:,6); end end
```

4 Random Walks and Markov Chains

A random walk on a directed graph consists of a sequence of vertices generated from a start vertex by probabilistically selecting an incident edge, traversing the edge to a new vertex, and repeating the process.

We generally assume the graph is *strongly connected*, meaning that for any pair of vertices x and y, the graph contains a path of directed edges starting at x and ending at y. If the graph is strongly connected, then, as we will see, no matter where the walk begins the fraction of time the walk spends at the different vertices of the graph converges to a stationary probability distribution.

Start a random walk at a vertex x and think of the starting probability distribution as putting a mass of one on x and zero on every other vertex. More generally, one could start with any probability distribution $\mathbf{p}$, where $\mathbf{p}$ is a row vector with nonnegative components summing to one, with p_x being the probability of starting at vertex x. The probability of being at vertex x at time $t + 1$ is the sum over each adjacent vertex y of being at y at time t and taking the transition from y to x. Let $\mathbf{p}(\mathbf{t})$ be a row vector with a component for each vertex specifying the probability mass of the vertex at time t and let $\mathbf{p}(\mathbf{t} + \mathbf{1})$ be the row vector of probabilities at time $t + 1$. In matrix notation[14]

$$\mathbf{p}(\mathbf{t})P = \mathbf{p}(\mathbf{t} + \mathbf{1})$$

where the ij^{th} entry of the matrix P is the probability of the walk at vertex i selecting the edge to vertex j.

A fundamental property of a random walk is that in the limit, the long-term average probability of being at a particular vertex is independent of the start vertex, or an initial probability distribution over vertices, provided only that the underlying graph is strongly connected. The limiting probabilities are called the *stationary probabilities*. This fundamental theorem is proved in the next section.

A special case of random walks, namely random walks on undirected graphs, has important connections to electrical networks. Here, each edge has a parameter called *conductance*, like electrical conductance. If the walk is at vertex x, it chooses an edge to traverse next from among all edges incident to x with probability proportional to its conductance. Certain basic quantities associated with random walks are hitting time, which is the expected time to reach vertex y starting at vertex x, and cover time, which is the expected time to visit every vertex. Qualitatively, for undirected graphs these quantities are all bounded above by polynomials in the number of vertices. The proofs of these facts will rely on the analogy between random walks and electrical networks.

[14] Probability vectors are represented by row vectors to simplify notation in equations like the one here.

random walk	Markov chain
graph	stochastic process
vertex	state
strongly connected	persistent
aperiodic	aperiodic
strongly connected and aperiodic	ergodic
undirected graph	time reversible

Table 5.1: Correspondence between terminology of random walks and Markov chains

Aspects of the theory of random walks were developed in computer science with a number of applications. Among others, these include defining the pagerank of pages on the World Wide Web by their stationary probability. An equivalent concept called a *Markov chain* had previously been developed in the statistical literature. A Markov chain has a finite set of *states*. For each pair of states x and y, there is a *transition probability* p_{xy} of going from state x to state y where for each x, $\sum_y p_{xy} = 1$. A random walk in the Markov chain starts at some state. At a given time step, if it is in state x, the next state y is selected randomly with probability p_{xy}. A Markov chain can be represented by a directed graph with a vertex representing each state and an edge with weight p_{xy} from vertex x to vertex y. We say that the Markov chain is *connected* if the underlying directed graph is strongly connected. That is, if there is a directed path from every vertex to every other vertex. The matrix P consisting of the p_{xy} is called the *transition probability matrix* of the chain. The terms "random walk" and "Markov chain" are used interchangeably. The correspondence between the terminologies of random walks and Markov chains is given in Table 5.1.

A state of a Markov chain is *persistent* if it has the property that should the state ever be reached, the random process will return to it with probability one. This is equivalent to the property that the state is in a strongly connected component with no out edges. For most of the chapter, we assume that the underlying directed graph is strongly connected. We discuss here briefly what might happen if we do not have strong connectivity. Consider the directed graph in Figure 4.1b with three strongly connected components, A, B, and C. Starting from any vertex in A, there is a nonzero probability of eventually reaching any vertex in A. However, the probability of returning to a vertex in A is less than one and thus vertices in A, and similarly vertices in B, are not persistent. From any vertex in C, the walk eventually will return with probability one to the vertex, since there is no way of leaving component C. Thus, vertices in C are persistent.

A connected Markov Chain is said to be *aperiodic* if the greatest common divisor of the lengths of directed cycles is one. It is known that for connected aperiodic chains, the

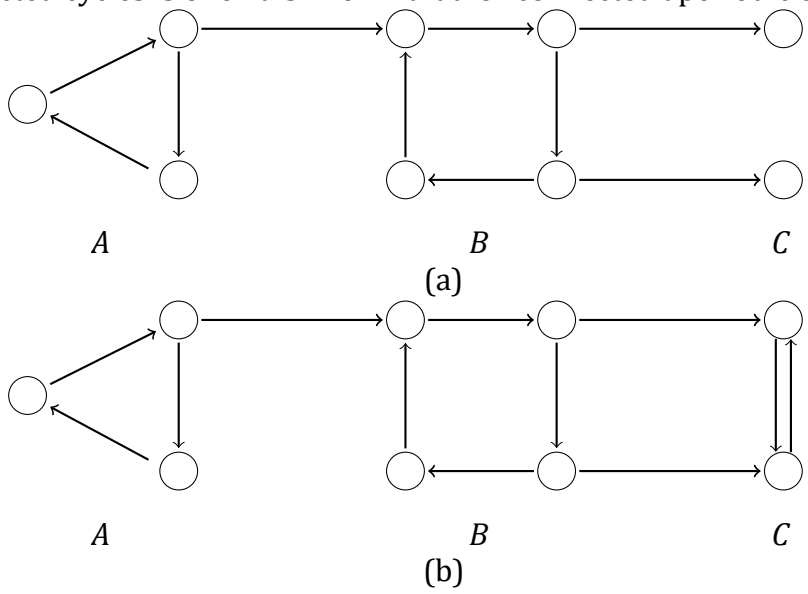


Figure 4.1: (a) A directed graph with vertices having no out out edges and a strongly connected component A with no in edges.
(b) A directed graph with three strongly connected components.

probability distribution of the random walk converges to a unique stationary distribution. Aperiodicity is a technical condition needed in this proof. Here, we do not prove this theorem and do not worry about aperiodicity at all. It turns out that if we take the average probability distribution of the random walk over the first t steps, then this average converges to a limiting distribution for connected chains (without assuming aperiodicity) and this average is what one uses in practice. We prove this limit theorem and explain its uses in what is called the Markov Chain Monte Carlo (MCMC) method.

Markov chains are used to model situations where all the information of the system necessary to predict the future can be encoded in the current state. A typical example is speech, where for a small k the current state encodes the last k syllables uttered by the speaker. Given the current state, there is a certain probability of each syllable being uttered next and these can be used to calculate the transition probabilities. Another example is a gambler's assets, which can be modeled as a Markov chain where the current state is the amount of money the gambler has on hand. The model would only be valid if the gambler's bets depend only on current assets, not the past history.

Later in the chapter, we study the widely used Markov Chain Monte Carlo method (MCMC). Here, the objective is to sample a large space according to some probability distribution p. The number of elements in the space may be very large, say 10^{100}. One

designs a Markov chain where states correspond to the elements of the space. The transition probabilities of the chain are designed so that the stationary probability of the chain is the probability distribution p with which we want to sample. One chooses samples by taking a random walk until the probability distribution is close to the stationary distribution of the chain and then selects the current state of the walk. The walk continues a number of steps until the probability distribution is nearly independent of where the walk was when the first element was selected. A second point is then selected, and so on. Although it is impossible to store the graph in a computer since it has 10^{100} vertices, to do the walk one needs only store the current vertex of the walk and be able to generate the adjacent vertices by some algorithm. What is critical is that the probability distribution of the walk converges to the stationary distribution in time logarithmic in the number of states.

We mention two motivating examples. The first is to select a point at random in d-space according to a probability density such as a Gaussian. Put down a grid and let each grid point be a state of the Markov chain. Given a probability density p, design transition probabilities of a Markov chain so that the stationary distribution is p. In general, the number of states grows exponentially in the dimension d, but if the time to converge to the stationary distribution grows polynomially in d, then one can do a random walk on the graph until convergence to the stationary probability. Once the stationary probability has been reached, one selects a point. To select a set of points, one must walk a number of steps between each selection so that the probability of the current point is independent of the previous point. By selecting a number of points one can estimate the probability of a region by observing the number of selected points in the region.

A second example is from physics. Consider an $n \times n$ grid in the plane with a particle at each grid point. Each particle has a spin of ±1. A configuration is a n^2 dimensional vector $\mathbf{v} = (v_1, v_2, \ldots, v_{n^2})$, where v_i is the spin of the i^{th} particle. There are 2^{n^2} spin configurations. The energy of a configuration is a function $f(\mathbf{v})$ of the configuration, not of any single spin. A central problem in statistical mechanics is to sample spin configurations according to their probability. It is easy to design a Markov chain with one state per spin configuration so that the stationary probability of a state is proportional to the state's energy. If a random walk gets close to the stationary probability in time polynomial in n rather than 2^{n^2}, then one can sample spin configurations according to their probability.

The Markov Chain has 2^{n^2} states, one per configuration. Two states in the Markov chain are adjacent if and only if the corresponding configurations $\mathbf{v}$ and $\mathbf{u}$ differ in just one coordinate ($u_i = v_i$ for all but one i). The Metropilis-Hastings random walk, described in more detail in Section 4.2, has a transition probability from a configuration $\mathbf{v}$ to an adjacent configuration $\mathbf{u}$ of

$$\frac{1}{n^2} \min\left(1, \frac{f(\mathbf{u})}{f(\mathbf{v})}\right).$$

As we will see, the Markov Chain has a stationary probability proportional to the energy. There are two more crucial facts about this chain. The first is that to execute a step in the chain, we do not need the whole chain, just the ratio $\frac{f(\mathbf{u})}{f(\mathbf{v})}$. The second is that under suitable assumptions, the chain approaches stationarity in time polynomial in *n.*

A quantity called the *mixing time*, loosely defined as the time needed to get close to the stationary distribution, is often much smaller than the number of states. In Section 4.4, we relate the mixing time to a combinatorial notion called *normalized conductance* and derive upper bounds on the mixing time in several cases.

4.1 Stationary Distribution

Let $\mathbf{p}(\mathbf{t})$ be the probability distribution after t steps of a random walk. Define the *long-term average probability distribution* $\mathbf{a}(\mathbf{t})$ by

$$\mathbf{a}(\mathbf{t}) = \frac{1}{t}\big(\mathbf{p}(0) + \mathbf{p}(1) + \cdots + \mathbf{p}(\mathbf{t}-1)\big).$$

The fundamental theorem of Markov chains asserts that for a connected Markov chain, $\mathbf{a}(\mathbf{t})$ converges to a limit probability vector $\mathbf{x}$ that satisfies the equations $\mathbf{x}P = \mathbf{x}$. Before proving the fundamental theorem of Markov chains, we first prove a technical lemma.

Lemma 4.1 *Let P be the transition probability matrix for a connected Markov chain. The $n\times(n+1)$ matrix $A = [P - I\ ,\ \mathbf{1}]$ obtained by augmenting the matrix $P - I$ with an additional column of ones has rank n.*

Proof: If the rank of $A = [P - I, \mathbf{1}]$ was less than n there would be a subspace of solutions to $A\mathbf{x} = \mathbf{0}$ of at least two-dimensions. Each row in P sums to one, so each row in $P - I$ sums to zero. Thus $\mathbf{x} = (\mathbf{1},0)$, where all but the last coordinate of $\mathbf{x}$ is 1, is one solution to $A\mathbf{x} = \mathbf{0}$. Assume there was a second solution $(\mathbf{x},\alpha)$ perpendicular to $(\mathbf{1},0)$. Then $(P - I)\mathbf{x}+\alpha\mathbf{1} = \mathbf{0}$ and for each i, $x_i = \sum_j p_{ij}x_j + \alpha$. Each x_i is a convex combination of some x_j plus α. Let S be the set of i for which x_i attains its maximum value. Since $\mathbf{x}$ is perpendicular to $\mathbf{1}$, some x_i is negative and thus $\bar{S}$ is not empty. Connectedness implies that some x_k of maximum value is adjacent to some x_l of lower value. Thus, $x_k > \sum_j p_{kj}x_j$. Therefore α must be greater than 0 in $x_k = \sum_j p_{kj}x_j + \alpha$..

On the other hand, the same argument with T the set of i with x_i taking its minimum value implies $\alpha < 0$. This contradiction falsifies the assumption of a second solution, thereby proving the lemma.■

Theorem 4.2 (Fundamental Theorem of Markov Chains) *For a connected Markov chain there is a unique probability vector π satisfying $\pi P = \pi$. Moreover, for any starting distribution,* $\lim \mathbf{a}(\mathbf{t})$ *exists and equals π.*

$t \to \infty$

Proof: Note that **a**(**t**) is itself a probability vector, since its components are nonnegative and sum to 1. Run one step of the Markov chain starting with distribution **a**(**t**); the distribution after the step is **a**(**t**)P. Calculate the change in probabilities due to this step. **a**

$$\begin{aligned}(\mathbf{t})P - \mathbf{a}(\mathbf{t}) &= \frac{1}{t}\left[\mathbf{p}(\mathbf{0})P + \mathbf{p}(\mathbf{1})P + \cdots + \mathbf{p}(\mathbf{t}-\mathbf{1})P\right] - \frac{1}{t}\left[\mathbf{p}(\mathbf{0}) + \mathbf{p}(\mathbf{1}) + \cdots + \mathbf{p}(\mathbf{t}-\mathbf{1})\right] \\ &= \frac{1}{t}\left[\mathbf{p}(\mathbf{1}) + \mathbf{p}(\mathbf{2}) + \cdots + \mathbf{p}(\mathbf{t})\right] - \frac{1}{t}\left[\mathbf{p}(\mathbf{0}) + \mathbf{p}(\mathbf{1}) + \cdots + \mathbf{p}(\mathbf{t}-\mathbf{1})\right] \\ &= \frac{1}{t}\left(\mathbf{p}(\mathbf{t}) - \mathbf{p}(\mathbf{0})\right).\end{aligned}$$

Thus, **b**(**t**) = **a**(**t**)P − **a**(**t**) satisfies $|\mathbf{b}(\mathbf{t})| \leq \frac{2}{t} \to 0, \text{ as } t \to \infty$.

By Lemma 4.1 above, $A = [P - I, \mathbf{1}]$ has rank n. The $n \times n$ submatrix B of A consisting of all its columns except the first is invertible. Let **c**(**t**) be obtained from **b**(**t**) by removing the first entry. Since **a**(**t**)P − **a**(**t**) = **b**(**t**) and B is obtained by deleting the first column of $P - I$ and adding a column of 1's, **a**(**t**)B = [**c**(**t**),1]. Then **a**(**t**) = [**c**(**t**) , 1]$B^{-1} \to$ [**0** , 1]B^{-1} establishing the theorem with π = [**0** , 1]B^{-1}. ■

We finish this section with the following lemma useful in establishing that a probability distribution is the stationary probability distribution for a random walk on a connected graph with edge probabilities.

Lemma 4.3 *For a random walk on a strongly connected graph with probabilities on the edges, if the vector π satisfies $\pi_x p_{xy} = \pi_y p_{yx}$ for all x and y and $\sum_x \pi_x = 1$, then π is the stationary distribution of the walk.*

Proof: Since π satisfies $\pi_x p_{xy} = \pi_y p_{yx}$, summing both sides, $\pi_x = \sum_y \pi_y p_{yx}$ and hence π satisfies $\pi = \pi P$. By Theorem 4.2, π is the unique stationary probability. ■

4.2 Markov Chain Monte Carlo

The Markov Chain Monte Carlo (MCMC) method is a technique for sampling a multivariate probability distribution $p(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2, \ldots, x_d)$. The MCMC method is used to estimate the expected value of a function $f(\mathbf{x})$

$$E(f) = \sum_{\mathbf{x}} f(\mathbf{x}) p(\mathbf{x}).$$

If each x_i can take on two or more values, then there are at least 2^d values for **x**, so an explicit summation requires exponential time. Instead, one could draw a set of samples,

where each sample $\mathbf{x}$ is selected with probability $p(\mathbf{x})$. Averaging f over these samples provides an estimate of the sum.

To sample according to $p(\mathbf{x})$, design a Markov Chain whose states correspond to the possible values of $\mathbf{x}$ and whose stationary probability distribution is $p(\mathbf{x})$. There are two general techniques to design such a Markov Chain: the Metropolis-Hastings algorithm and Gibbs sampling, which we will describe in the next two subsections. The Fundamental Theorem of Markov Chains, Theorem 4.2, states that the average of the function f over states seen in a sufficiently long run is a good estimate of $E(f)$. The harder task is to show that the number of steps needed before the long-run average probabilities are close to the stationary distribution grows polynomially in d, though the total number of states may grow exponentially in d. This phenomenon known as *rapid mixing* happens for a number of interesting examples. Section 4.4 presents a crucial tool used to show rapid mixing.

We used $\mathbf{x} \in \mathbf{R}^d$ to emphasize that distributions are multi-variate. From a Markov chain perspective, each value $\mathbf{x}$ can take on is a state, i.e., a vertex of the graph on which the random walk takes place. Henceforth, we will use the subscripts *i,j,k,...* to denote states and will use p_i instead of $p(x_1,x_2,...,x_d)$ to denote the probability of the state corresponding to a given set of values for the variables. Recall that in the Markov chain terminology, vertices of the graph are called states.

Recall the notation that $\mathbf{p}(\mathbf{t})$ is the row vector of probabilities of the random walk being at each state (vertex of the graph) at time t. So, $\mathbf{p}(\mathbf{t})$ has as many components as there are states and its i^{th} component is the probability of being in state i at time t. Recall the long-term t-step average is

$$\mathbf{a}(\mathbf{t}) = \frac{1}{t}\left[\mathbf{p}(\mathbf{0}) + \mathbf{p}(\mathbf{1}) + \cdots + \mathbf{p}(\mathbf{t}-\mathbf{1})\right]. \tag{4.1}$$

The expected value of the function f under the probability distribution $\mathbf{p}$ is $E(f) = \sum_i f_i p_i$ where f_i is the value of f at state i. Our estimate of this quantity will be the average value of f at the states seen in a t step walk. Call this estimate γ. Clearly, the expected value of γ is

$$E(\gamma) = \sum_i f_i \left(\frac{1}{t} \sum_{j=1}^{t} \text{Prob walk is in state } i \text{ at time } j) \right) = \sum_i f_i a_i(t).$$

The expectation here is with respect to the "coin tosses" of the algorithm, not with respect to the underlying distribution $\mathbf{p}$. Let $f_{\max}$ denote the maximum absolute value of f. It is easy to see that

$$\left| \sum_i f_i p_i - E(\gamma) \right| \leq f_{\max} \sum_i |p_i - a_i(t)| = f_{\max} ||\mathbf{p} - \mathbf{a}(\mathbf{t})||_1 \tag{4.2}$$

where the quantity $||\mathbf{p}-\mathbf{a}(\mathbf{t})||_1$ is the l_1 distance between the probability distributions $\mathbf{p}$ and $\mathbf{a}(\mathbf{t})$, often called the "total variation distance" between the distributions. We will build tools to upper bound $||\mathbf{p} - \mathbf{a}(\mathbf{t})||_1$. Since $\mathbf{p}$ is the stationary distribution, the t for which $||\mathbf{p}-\mathbf{a}(\mathbf{t})||_1$ becomes small is determined by the rate of convergence of the Markov chain to its steady state.

The following proposition is often useful.

Proposition 4.4 *For two probability distributions* $\mathbf{p}$ *and* $\mathbf{q}$,

$$||\mathbf{p} - \mathbf{q}||_1 = \sum_i (p_i - q_i)^+ = \sum_i (q_i - p_i)^+$$

where $x^+ = x$ *if* $x \geq 0$ *and* $x^+ = 0$ *if* $x < 0$.

The proof is left as an exercise.

4.2.1 Metropolis-Hasting Algorithm

The Metropolis-Hasting algorithm is a general method to design a Markov chain whose stationary distribution is a given target distribution $\mathbf{p}$. Start with a connected undirected graph G on the set of states. If the states are the lattice points $(x_1,x_2,\ldots,x_d)$ in $\mathbf{R}^d$ with $x_i \in \{0,1,2,,,\ldots,n\}$, then G could be the lattice graph with $2d$ coordinate edges at each interior vertex. In general, let r be the maximum degree of any vertex of G. The transitions of the Markov chain are defined as follows. At state i select neighbor j with probability $\frac{1}{r}$. Since the degree of i may be less than r, with some probability no edge is selected and the walk remains at i. If a neighbor j is selected and $p_j \geq p_i$, go to j. If $p_j < p_i$, go to j with probability p_j/p_i and stay at i with probability $1 - \frac{p_j}{p_i}$. Intuitively, this favors "heavier" states with higher p_i values. For i adjacent to j in G,

$$p_{ij} = \frac{1}{r} \min\left(1, \frac{p_j}{p_i}\right)$$

and

$$p_{ii} = 1 - \sum_{j \neq i} p_{ij}.$$

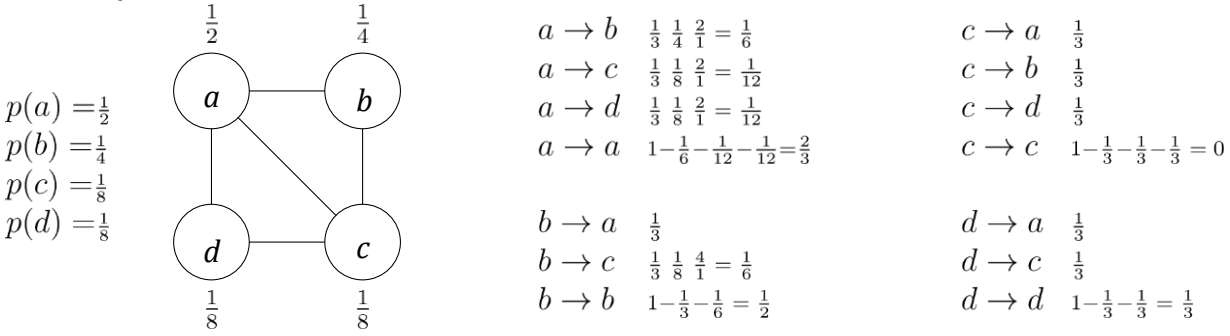
Thus,

$$p_i p_{ij} = \frac{p_i}{r} \min\left(1, \frac{p_j}{p_i}\right) = \frac{1}{r} \min(p_i, p_j) = \frac{p_j}{r} \min\left(1, \frac{p_i}{p_j}\right) = p_j p_{ji}.$$

By Lemma 4.3, the stationary probabilities are indeed p_i as desired.

Example: Consider the graph in Figure 4.2. Using the Metropolis-Hasting algorithm, assign transition probabilities so that the stationary probability of a random walk is

$p(a) = \frac{1}{2}$, $p(b) = \frac{1}{4}$, $p(c) = \frac{1}{8}$, and $p(d) = \frac{1}{8}$. The maximum degree of any vertex is three, so at *a,* the probability of taking the edge (a, b) is $\frac{1}{3}\frac{1}{4}\frac{2}{1}$ or $\frac{1}{6}$. The probability of taking the edge (a, c) is $\frac{1}{3}\frac{1}{8}\frac{2}{1}$ or $\frac{1}{12}$ and of taking the edge (a, d) is $\frac{1}{3}\frac{1}{8}\frac{2}{1}$ or $\frac{1}{12}$. Thus, the probability of staying at a is $\frac{2}{3}$. The probability of taking the edge from b to a is $\frac{1}{3}$. The probability of taking the edge from c to a is $\frac{1}{3}$ and the probability of taking the edge from *d* to *a* is $\frac{1}{3}$. Thus, the stationary probability of *a* is $\frac{1}{4}\frac{1}{3} + \frac{1}{8}\frac{1}{3} + \frac{1}{8}\frac{1}{3} + \frac{1}{2}\frac{2}{3} = \frac{1}{2}$, which is the desired probability. ■



$$p(a) = p(a)p(a \to a) + p(b)p(b \to a) + p(c)p(c \to a) + p(d)p(d \to a)$$
$$= \frac{1}{2}\frac{2}{3} + \frac{1}{4}\frac{1}{3} + \frac{1}{8}\frac{1}{3} + \frac{1}{8}\frac{1}{3} = \frac{1}{2}$$

$$p(b) = p(a)p(a \to b) + p(b)p(b \to b) + p(c)p(c \to b)$$
$$= \frac{1}{2}\frac{1}{6} + \frac{1}{4}\frac{1}{2} + \frac{1}{8}\frac{1}{3} = \frac{1}{4}$$

$$p(c) = p(a)p(a \to c) + p(b)p(b \to c) + p(c)p(c \to c) + p(d)p(d \to c)$$
$$= \frac{1}{2}\frac{1}{12} + \frac{1}{4}\frac{1}{6} + \frac{1}{8}0 + \frac{1}{8}\frac{1}{3} = \frac{1}{8}$$

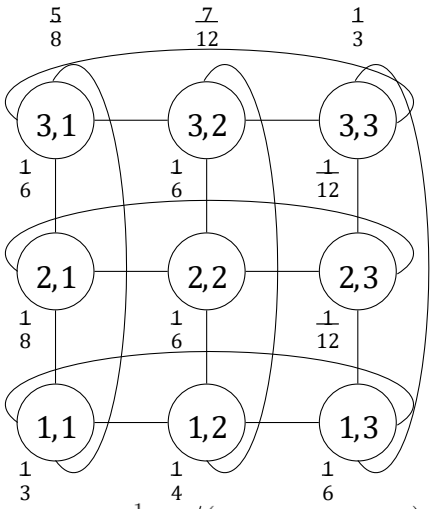
$$p(d) = p(a)p(a \to d) + p(c)p(c \to d) + p(d)p(d \to d)$$
$$= \frac{1}{2}\frac{1}{12} + \frac{1}{8}\frac{1}{3} + \frac{1}{8}\frac{1}{3} = \frac{1}{8}$$

Figure 4.2: Using the Metropolis-Hasting algorithm to set probabilities for a random walk so that the stationary probability will be the desired probability.

4.2.2 Gibbs Sampling

Gibbs sampling is another Markov Chain Monte Carlo method to sample from a multivariate probability distribution. Let $p(\mathbf{x})$ be the target distribution where $\mathbf{x} = (x_1,\ldots,x_d)$. Gibbs sampling consists of a random walk on an undirectd graph whose vertices correspond to the values of $\mathbf{x} = (x_1,\ldots,x_d)$ and in which there is an edge from **x** to **y** if **x** and **y** differ in only one coordinate. Thus, the underlying graph is like a *d*-dimensional lattice except that the vertices in the same coordinate line form a clique.

To generate samples of $\mathbf{x} = (x_1,\ldots,x_d)$ with a target distribution $p(\mathbf{x})$, the Gibbs sampling algorithm repeats the following steps. One of the variables x_i is chosen to be updated. Its new value is chosen based on the marginal probability of x_i with the other variables fixed. There are two commonly used schemes to determine which x_i to update. One scheme is to choose x_i randomly, the other is to choose x_i by sequentially scanning from x_1 to x_d.



Suppose that $\mathbf{x}$ and $\mathbf{y}$ are two states that differ in only one coordinate. Without loss of generality let that coordinate be the first. Then, in the scheme where a coordinate is randomly chosen to modify, the probability $p_{\mathbf{xy}}$ of going from $\mathbf{x}$ to $\mathbf{y}$ is

$p(1,1) = \frac{5}{12}$ $p(1,2) = \frac{1}{6}$ $p(1,3) = p(2,1) = \frac{3}{8}$

$p(2,2) = p(2,3) = \frac{1}{8}$ $= p(3,1) = \frac{3}{4}$ $p(3,2) = \frac{1}{3}$

$p(3,3) = \frac{1}{4}$

$\frac{1}{6}$

$\frac{1}{12}$

$\frac{1}{6}$

$\frac{1}{6}$

$\frac{1}{12}$

$$p_{(11)(12)} = \tfrac{1}{d}p_{12}/(p_{11}+p_{12}+p_{13}) = \tfrac{1}{2}\ (\tfrac{1}{4})/(\tfrac{1}{3}\ \tfrac{1}{4}\ \tfrac{1}{6}) = \tfrac{1}{8}/\tfrac{9}{12} = \tfrac{1}{8}\ \tfrac{4}{3} = \tfrac{1}{6}$$

Calculation of edge probability $p_{(11)(12)}$

$$p_{(11)(12)} = \tfrac{1}{2}\ \tfrac{1}{4}\ \tfrac{4}{3} = \tfrac{1}{6} \quad p_{(12)(11)} = \tfrac{1}{2}\ \tfrac{1}{3}\ \tfrac{4}{3} = \tfrac{2}{9} \quad p_{(13)(11)} = \tfrac{1}{2}\ \tfrac{1}{3}\ \tfrac{4}{3} = \tfrac{2}{9} \quad p_{(21)(22)} = \tfrac{1}{2}\tfrac{1}{6}\tfrac{8}{3} = \tfrac{2}{9}$$
$$p_{(11)(13)} = \tfrac{1}{2}\ \tfrac{1}{6}\ \tfrac{4}{3} = \tfrac{1}{9} \quad p_{(12)(13)} = \tfrac{1}{2}\ \tfrac{1}{6}\ \tfrac{4}{3} = \tfrac{1}{9} \quad p_{(13)(12)} = \tfrac{1}{2}\ \tfrac{1}{4}\ \tfrac{4}{3} = \tfrac{1}{6} \quad p_{(21)(23)} = \tfrac{1}{2}\tfrac{1}{12}\tfrac{8}{3} = \tfrac{1}{9}$$
$$p_{(11)(21)} = \tfrac{1}{2}\ \tfrac{1}{8}\ \tfrac{8}{5} = \tfrac{1}{10} \quad p_{(12)(22)} = \tfrac{1}{2}\ \tfrac{1}{6}\ \tfrac{12}{7} = \tfrac{1}{7} \quad p_{(13)(23)} = \tfrac{1}{2}\ \tfrac{1}{12}\ \tfrac{3}{1} = \tfrac{1}{8} \quad p_{(21)(11)} = \tfrac{1}{2}\tfrac{1}{3}\tfrac{8}{5} = \tfrac{4}{15}$$
$$p_{(11)(31)} = \tfrac{1}{2}\ \tfrac{1}{6}\ \tfrac{8}{5} = \tfrac{2}{15} \quad p_{(12)(32)} = \tfrac{1}{2}\ \tfrac{1}{6}\ \tfrac{12}{7} = \tfrac{1}{7} \quad p_{(13)(33)} = \tfrac{1}{2}\ \tfrac{1}{12}\ \tfrac{3}{1} = \tfrac{1}{8} \quad p_{(21)(31)} = \tfrac{1}{2}\tfrac{1}{6}\tfrac{8}{5} = \tfrac{2}{15}$$

Edge probabilities.

$$p_{11}p_{(11)(12)} = \tfrac{1}{3}\ \tfrac{1}{6} = \tfrac{1}{4}\ \tfrac{2}{9} = p_{12}p_{(12)(11)}$$
$$p_{11}p_{(11)(13)} = \tfrac{1}{3}\ \tfrac{1}{9} = \tfrac{1}{6}\ \tfrac{2}{9} = p_{13}p_{(13)(11)}$$
$$p_{11}p_{(11)(21)} = \tfrac{1}{3}\ \tfrac{1}{10} = \tfrac{1}{8}\ \tfrac{4}{15} = p_{21}p_{(21)(11)}$$

Verification of a few edges, $p_i p_{ij} = p_j p_{ji}$.

Note that the edge probabilities out of a state such as (1,1) do not add up to one. That is, with some probability the walk stays at the state that it is in. For example,
$p_{(11)(11)} = 1 - (p_{(11)(12)} + p_{(11)(13)} + p_{(11)(21)} + p_{(11)(31)}) = 1 - \frac{1}{6} - \frac{1}{24} - \frac{1}{32} - \frac{1}{24} = \frac{9}{32}$.

Figure 4.3: Using the Gibbs algorithm to set probabilities for a random walk so that the stationary probability will be a desired probability.

$$p_{\mathbf{xy}} = \frac{1}{d} p(y_1 | x_2, x_3, \ldots, x_d).$$

The normalizing constant is $1/d$ since $\sum_{y_1} p(y_1|x_2,x_3,\ldots,x_d)$ equals 1 and summing over d coordinates

$$\sum_{i=1}^{d} p(y_i|x_1,x_2,\ldots,x_{i-1},x_{i+1}\ldots x_d) = d$$

gives a value of d. Similarly,

$$\begin{aligned} p_{\mathbf{yx}} &= \frac{1}{d} p(x_1 | y_2, y_3, \ldots, y_d) \\ &= \frac{1}{d} p(x_1 | x_2, x_3, \ldots, x_d). \end{aligned}$$

Here use was made of the fact that for $j \neq 1$, $x_j = y_j$.

It is simple to see that this chain has stationary probability proportional to $p(\mathbf{x})$. Rewrite $p_{\mathbf{xy}}$ as

$$\begin{aligned} p_{\mathbf{xy}} &= \frac{1}{d} \frac{p(y_1 | x_2, x_3, \ldots, x_d) p(x_2, x_3, \ldots, x_d)}{p(x_2, x_3, \ldots, x_d)} \\ &= \frac{1}{d} \frac{p(y_1, x_2, x_3, \ldots, x_d)}{p(x_2, x_3, \ldots, x_d)} \\ &= \frac{1}{d} \frac{p(\mathbf{y})}{p(x_2, x_3, \ldots, x_d)} \end{aligned}$$

again using $x_j = y_j$ for $j \neq 1$. Similarly write

$$p_{\mathbf{yx}} = \frac{1}{d} \frac{p(\mathbf{x})}{p(x_2, x_3, \ldots, x_d)}$$

from which it follows that $p(\mathbf{x})p_{xy} = p(\mathbf{y})p_{yx}$. By Lemma 4.3 the stationary probability of the random walk is $p(\mathbf{x})$.

4.3 Areas and Volumes

Computing areas and volumes is a classical problem. For many regular figures in two and three dimensions there are closed form formulae. In Chapter 2, we saw how to compute volume of a high dimensional sphere by integration. For general convex sets in d-space, there are no closed form formulae. Can we estimate volumes of d-dimensional convex sets in time that grows as a polynomial function of d? The MCMC method answes this question in the affirmative.

One way to estimate the area of the region is to enclose it in a rectangle and estimate the ratio of the area of the region to the area of the rectangle by picking random points in the rectangle and seeing what proportion land in the region. Such methods fail in high dimensions. Even for a sphere in high dimension, a cube enclosing the sphere has exponentially larger area, so exponentially many samples are required to estimate the volume of the sphere.

It turns out, however, that the problem of estimating volumes of sets can be reduced to the problem of drawing uniform random samples from sets. Suppose one wants to estimate the volume of a convex set R. Create a concentric series of larger and larger spheres[15] $S_1, S_2, \ldots, S_k$ such that S_1 is contained in R and S_k contains R. Then

$$\mathrm{Vol}(R) = \mathrm{Vol}(S_k \cap R) = \frac{\mathrm{Vol}(S_k \cap R)}{\mathrm{Vol}(S_{k-1} \cap R)} \frac{\mathrm{Vol}(S_{k-1} \cap R)}{\mathrm{Vol}(S_{k-2} \cap R)} \cdots \frac{\mathrm{Vol}(S_2 \cap R)}{\mathrm{Vol}(S_1 \cap R)} \mathrm{Vol}(S_1)$$

If the radius of the sphere S_i is $1 + \frac{1}{d}$ times the radius of the sphere S_{i-1}, then we have:

$$1 \leq \frac{\mathrm{Vol}(S_i \cap R)}{\mathrm{Vol}(S_{i-1} \cap R)} \leq e$$

because $\mathrm{Vol}(S_i)/\mathrm{Vol}(S_{i-1}) = \left(1 + \frac{1}{d}\right)^d < e$, and the fraction of S_i occupied by R is less than or equal to the fraction of S_{i-1} occupied by R (due to the convexity of R and the fact that the center of the spheres lies in R). This implies that the ratio $\frac{Vol(S_i \cap R)}{Vol(S_{i-1} \cap R)}$ can be estimated by rejection sampling, i.e., selecting points in $S_i \cap R$ uniformly at random and computing the fraction in $S_{i-1} \cap R$, provided one can select points at random from a d-dimensional convex region.

The number of spheres is at most

$$O(\log_{1+(1/d)} r) = O(rd)$$

where r is the ratio of the radius of S_k to the radius of S_1. This means that it suffices to estimate each ratio to a factor of $(1 \pm \frac{\epsilon}{erd})$ in order to estimate the overall volume to error $1 \pm \epsilon$.

It remains to show how to draw a uniform random sample from a d-dimensional convex set. Here we will use the convexity of the set R and thus the sets $S_i \cap R$ so that the Markov chain technique will converge quickly to its stationary probability. To select a random sample from a d-dimensional convex set, impose a grid on the region and do a

[15] One could also use rectangles instead of spheres.

random walk on the grid points. At each time, pick one of the $2d$ coordinate neighbors of the current grid point, each with probability $1/(2d)$ and go to the neighbor if it is still in the

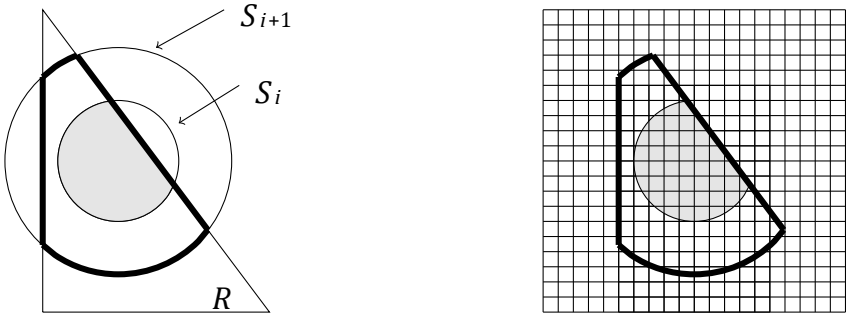


Figure 4.4: By sampling the area inside the dark line and determining the fraction of points in the shaded region we compute $\frac{Vol(S_{i+1} \cap R)}{Vol(S_i \cap R)}$.
To sample we create a grid and assign a probability of one to each grid point inside the dark lines and zero outside. Using Metropolis-Hasting edge probabilities the stationary probability will be uniform for each point inside the region and we can sample points uniformly and determine the fraction within the shaded region.

set; otherwise, stay put and repeat. If the grid length in each of the d coordinate directions is at most some a, the total number of grid points in the set is at most a^d. Although this is exponential in d, the Markov chain turns out to be rapidly mixing (the proof is beyond our scope here) and leads to polynomial time bounded algorithm to estimate the volume of any convex set in $\mathbf{R}^d$.

4.4 Convergence of Random Walks on Undirected Graphs

The Metropolis-Hasting algorithm and Gibbs sampling both involve random walks on edge-weighted undirected graphs. Given an edge-weighted undirected graph, let w_{xy} denote the weight of the edge between nodes x and y, with $w_{xy} = 0$ if no such edge exists. Let $w_x = \sum_y w_{xy}$. The Markov chain has transition probabilities $p_{xy} = w_{xy}/w_x$. We assume the chain is connected.

We now claim that the stationary distribution π of this walk has π_x proportional to w_x, i.e., $\pi_x = w_x/w_{total}$ for $w_{total} = \sum_{x0} w_{x0}$. Specifically, notice that

$$w_x p_{xy} = w_x \frac{w_{xy}}{w_x} = w_{xy} = w_{yx} = w_y \frac{w_{yx}}{w_y} = w_y p_{yx}.$$

Therefore $(w_x/w_{total})p_{xy} = (w_y/w_{total})p_{yx}$ and Lemma 4.3 implies that the values $\pi_x = w_x/w_{total}$ are the stationary probabilities.

An important question is how fast the walk starts to reflect the stationary probability of the Markov process. If the convergence time was proportional to the number of states, algorithms such as Metropolis-Hasting and Gibbs sampling would not be very useful since

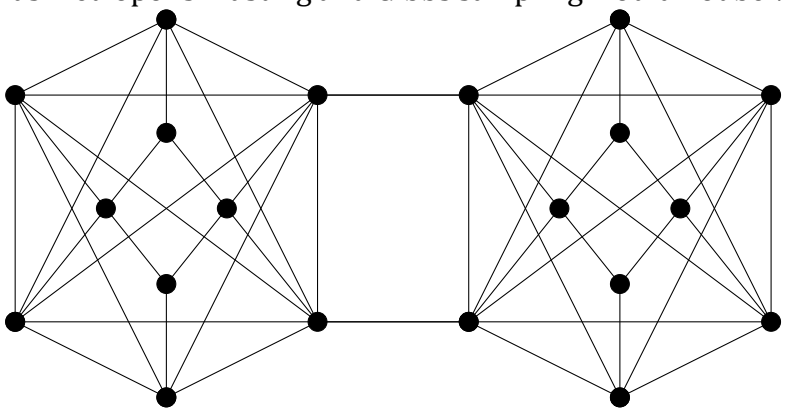


Figure 4.5: A network with a constriction. All edges have weight 1.

the number of states can be exponentially large.

There are clear examples of connected chains that take a long time to converge. A chain with a constriction, see Figure 4.5, takes a long time to converge since the walk is unlikely to cross the narrow passage between the two halves, both of which are reasonably big. We will show in Theorem 4.5 that the time to converge is quantitatively related to the tightest constriction.

We define below a combinatorial measure of constriction for a Markov chain, called the *normalized conductance*. We will relate normalized conductance to the time by which the average probability distribution of the chain is guaranteed to be close to the stationary probability distribution. We call this ε-mixing time:

Definition 4.1 *Fix $\varepsilon > 0$. The* ε-mixing time *of a Markov chain is the minimum integer t such that for any starting distribution* **p**, *the 1-norm difference between the t-step running average probability distribution*[16] *and the stationary distribution is at most ε.* ■

Definition 4.2 *For a subset S of vertices, let $\pi(S)$ denote $\sum_{x \in S} \pi_x$. The normalized conductance $\Phi(S)$ of S is*

$$\Phi(S) = \frac{\sum_{(x,y)\in(S,\bar{S})} \pi_x p_{xy}}{\min\left(\pi(S), \pi(\bar{S})\right)}.$$

[16] Recall that $\mathbf{a}(\mathbf{t}) = \frac{1}{t}\left(\mathbf{p}(\mathbf{0}) + \mathbf{p}(\mathbf{1}) + \cdots + \mathbf{p}(\mathbf{t-1})\right)$ is called the running average distribution.

∎

There is a simple interpretation of $\Phi(S)$. Suppose without loss of generality that $\pi(S) \leq \pi(\bar{S})$. Then, we may write $\Phi(S)$ as

$$\Phi(S) = \sum_{x \in S} \underbrace{\frac{\pi_x}{\pi(S)}}_{a} \underbrace{\sum_{y \in \bar{S}} p_{xy}}_{b}.$$

Here, a is the probability of being in x if we were in the stationary distribution restricted to S and b is the probability of stepping from x to $\bar{S}$ in a single step. Thus, $\Phi(S)$ is the probability of moving from S to $\bar{S}$ in one step if we are in the stationary distribution restricted to S.

It is easy to show that if we started in the distribution $p_{0,x} = \pi_s/\pi(S)$ for $x \in S$ and $p_{0,x} = 0$ for $x \in \bar{S}$, the expected number of steps before we step into $\bar{S}$ is

$$1\Phi(S) + 2(1 - \Phi(S))\Phi(S) + 3(1 - \Phi(S))^2\Phi(S) + \cdots = \frac{1}{\Phi(S)}.$$

Clearly, to be close to the stationary distribution, we must at least get to $\bar{S}$ once. So, mixing time is lower bounded by $1/\Phi(S)$. Since we could have taken any S, mixing time is lower bounded by the minimum over all S of $\Phi(S)$. We define this quantity to be the normalized conductance of the Markov Chain.

Definition 4.3 *The* normalized conductance *of the Markov chain, denoted* Φ*, is defined by*

$$\Phi = \min_{S \subset V, S \neq \{\}} \Phi(S).$$

As we just argued, normalized conductance being high is a necessary condition for rapid mixing. The theorem below proves the converse that normalized conductance being high is sufficient for mixing. Intuitively, if Φ is large, the walk rapidly leaves any subset of states. But the proof of the theorem is quite difficult. After we prove it, we will see examples where the mixing time is much smaller than the cover time. That is, the number of steps before a random walk reaches a random state independent of its starting state is much smaller than the average number of steps needed to reach every state. In fact for some graphs, called expanders, the mixing time is logarithmic in the number of states.

Theorem 4.5 *The ε-mixing time of a random walk on an undirected graph is*

$$O\left(\frac{\ln(1/\pi_{min})}{\Phi^2 \varepsilon^3}\right)$$

where π_{min} *is the minimum stationary probability of any state.*

Proof: Let $t = \frac{c \ln(1/\pi_{\min})}{\Phi^2 \varepsilon^3}$, for a suitable constant c. Let

$$\mathbf{a} = \mathbf{a}(t) = \frac{1}{t}(\mathbf{p}(0) + \mathbf{p}(1) + \cdots + \mathbf{p}(t-1))$$

be the running average distribution. We need to show that $||\mathbf{a} - \pi||_1 \leq \varepsilon$. Let

$$v_i = \frac{a_i}{\pi_i},$$

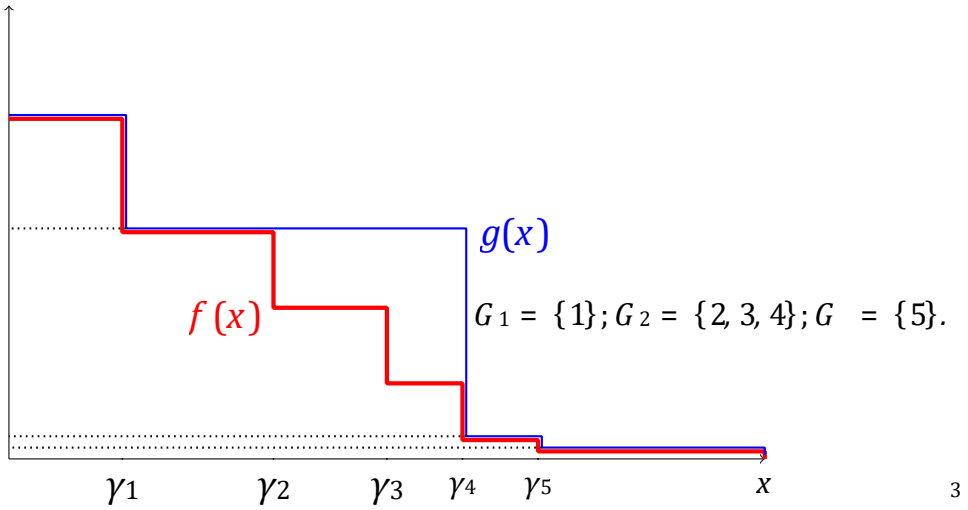


Figure 4.6: Bounding l_1 distance.

and renumber states so that $v_1 \geq v_2 \geq v_3 \geq \cdots$. Thus, early indices i for which $v_i > 1$ are states that currently have too much probability, and late indices i for which $v_i < 1$ are states that currently have too little probability.

Intuitively, to show that $||\mathbf{a} - \pi||_1 \leq \varepsilon$ it is enough to show that the values v_i are relatively flat and do not drop too fast as we increase i. We begin by reducing our goal to a formal statement of that form. Then, in the second part of the proof, we prove that v_i do not fall fast using the concept of "probability flows".

We call a state i for which $v_i > 1$ "heavy" since it has more probability according to $\mathbf{a}$ than its stationary probability. Let i_0 be the maximum i such that $v_i > 1$; it is the last heavy state. By Proposition (4.4):

$$||\mathbf{a} - \pi||_1 = 2\sum_{i=1}^{i_0}(v_i - 1)\pi_i = 2\sum_{i \geq i_0+1}(1 - v_i)\pi_i. \tag{4.3}$$

Let $\gamma_i = \pi_1 + \pi_2 + \cdots + \pi_i$.

Define a function f: $[0,\gamma_{i_0}] \to <$ by $f(x) = v_i - 1$ for $x \in [\gamma_{i-1},\gamma_i)$. See Figure 4.6. Now,

$$\sum_{i=1}^{i_0}(v_i - 1)\pi_i = \int_0^{\gamma_{i_0}} f(x)\, dx. \tag{4.4}$$

We make one more technical modification. We divide $\{1,2,\ldots,i_0\}$ into groups $G_1,G_2,G_3,\ldots,G_r$, of contiguous subsets. We specify the groups later. Let $u_t = \text{Max}_{i\in G_t} v_i$ be the maximum value of v_i within G_t. Define a new function $g(x)$ by $g(x) = u_t - 1$ for $x \in \cup_{i\in G_t}[\gamma_{i-1},\gamma_i)$; see Figure 4.6. Since $g(x) \geq f(x)$

$$\int_0^{\gamma_{i_0}} f(x)\, dx \leq \int_0^{\gamma_{i_0}} g(x)\, dx. \tag{4.5}$$

We now assert (with $u_{r+1} = 1$):

$$\int_0^{\gamma_{i_0}} g(x)\, dx = \sum_{t=1}^{r} \pi(G_1 \cup G_2 \cup \ldots \cup G_t)(u_t - u_{t+1}). \tag{4.6}$$

This is just the statement that the area under $g(x)$ in the figure is exactly covered by the rectangles whose bottom sides are the dotted lines. We leave the formal proof of this to the reader. We now focus on proving that

$$\sum_{t=1}^{r} \pi(G_1 \cup G_2 \cup \ldots \cup G_t)(u_t - u_{t+1}) \leq \varepsilon/2, \tag{4.7}$$

for a sub-division into groups we specify which suffices by 4.3, 4.4, 4.5 and 4.6. While we start the proof of (4.7) with a technical observation (4.8), its proof will involve two nice ideas: the notion of probability flow and reckoning probability flow in two different ways.

First, the technical observation: if $\sum_{i\geq i_0+1}(1-v_i)\pi_i \leq \varepsilon$ then we would be done by (4.3).

So assume now that $\sum_{i\geq i_0+1}(1 - v_i)\pi_i > \varepsilon/2$ from which it follows that $\sum_{i\geq i_0+1} \pi_i \geq \varepsilon/2$ and so, for any subset A of heavy nodes,

$$\text{Min}(\pi(A), \pi(\bar{A})) \geq \frac{\varepsilon}{2}\pi(A). \tag{4.8}$$

We now define the subsets. G_1 will be just $\{1\}$. In general, suppose $G_1,G_2,\ldots,G_{t-1}$ have already been defined. We start G_t at $i_t = 1+$ (end of G_{t-1}). Let $i_t = k$. We will define l, the last element of G_t to be the largest integer greater than or equal to k and at most i_0 so that

$$\sum_{j=k+1}^{l} \pi_j \leq \frac{\varepsilon\Phi\gamma_k}{4}.$$

In Lemma 4.6 which follows this theorem prove that for groups $G_1,G_2,\ldots,G_r,u_1.u_2,\ldots,u_r,u_{r+1}$ as above

$$\pi(G_1 \cup G_2 \cup \ldots G_r)(u_t - u_{t+1}) \leq \frac{8}{t\Phi\varepsilon.}$$

Now to prove (4.7), we only need an upper bound on r, the number of groups. If $G_t = \{k,k+1,\ldots,l\}$, with $l < i_0$, then by definition of l, we have $\gamma_{l+1} \geq (1 + \frac{\varepsilon\Phi}{2})\gamma_k$. So, $r \leq \ln_{1+(\varepsilon\Phi/2)}(1/\pi_1)+2 \leq \ln(1/\pi_1)/(\varepsilon\Phi/2)+2$. This completes the proof of (4.7) and the theorem. ■

We complete the proof of Theorem 4.5 with the proof of Lemma 4.6. The notation in the lemma is that from the theorem.

Lemma 4.6 *Suppose groups $G_1,G_2,\ldots,G_r,u_1.u_2,\ldots,u_r,u_{r+1}$ are as above. Then,*

$$\pi(G_1 \cup G_2 \cup \ldots G_r)(u_t - u_{t+1}) \leq \frac{8}{t\Phi\varepsilon.}$$

Proof: This is the main lemma. The proof of the lemma uses a crucial idea of probability flows. We will use two ways of calculating the probability flow from heavy states to light states when we execute one step of the Markov chain starting at probabilities **a**. The probability vector after that step is **a***P*. Now, **a** − **a***P* is the net loss of probability for each state due to the step.

Consider a particular group $G_t = \{k,k+1,\ldots,l\}$, say. First consider the case when $k < i_0$. Let $A = \{1,2,\ldots,k\}$. The net loss of probability for each state from the set A in one step is $\sum_{i=1}^{k}(a_i - (\mathbf{a}P)_i)$ which is at most $\frac{2}{t}$ by the proof of Theorem 4.2.

Another way to reckon the net loss of probability from A is to take the difference of the probability flow from A to A^- and the flow from A^- to A. For any $i < j$, net-flow(i,j) = flow(i,j) − flow(j,i) = $\pi_i p_{ij} v_i - \pi_j p_{ji} v_j = \pi_j p_{ji}(v_i - v_j) \geq 0$,

Thus, for any two states i and j, with i heavier than j, i.e., $i < j$, there is a non-negative net flow from i to j. (This is intuitively reasonable since it says that probability is flowing from heavy to light states.) Since $l \geq k$, the flow from A to $\{k+1,k+2,\ldots,l\}$ minus the flow from $\{k+1,k+2,\ldots,l\}$ to A is nonnegative. Since for $i \leq k$ and $j > l$, we have $v_i \geq v_k$ and $v_j \leq v_{l+1}$, the net loss from A is at least

$$\sum_{i\leq k} \sum_{i\leq k\, j>l} \pi_j p_{ji}(v_i - v_j) \geq (v_k - v_{l+1}) \sum_{j>l} \pi_j p_{ji}.$$

Thus,

$$(v_k - v_{l+1}) \sum_{\substack{i\leq k \\ j>l}} \pi_j p_{ji} \leq \frac{2}{t} \quad . \tag{4.9}$$

Since

$$\sum_{k} \sum_{l} \pi_j p_{ji} \leq \sum_{l} \pi_j \leq \varepsilon\Phi\pi(A)/4$$

$i=1\ j=k+1$ $j=k+1$

and by the definition of Φ, using (4.8)

$$\sum_{i\le k<j} \pi_j p_{ji} \ge \Phi \mathrm{Min}(\pi(A),\pi(\bar{A})) \ge \varepsilon\Phi\gamma_k/2,$$

$$\sum_{\substack{i\le k\\ j>l}} \pi_j p_{ji} = \sum_{i\le k<j} \pi_j p_{ji} - \sum_{i\le k;j\le l} \pi_j p_{ji} \ge \varepsilon\Phi\gamma_k/4$$

we have, . Substituting this into the inequality (4.9) gives

$$v_k - v_{l+1} \le \frac{8}{t\varepsilon\Phi\gamma_k}, \tag{4.10}$$

proving the lemma provided $k < i_0$. If $k = i_0$, the proof is similar but simpler. ∎

4.4.1 Using Normalized Conductance to Prove Convergence

We now apply Theorem 4.5 to some examples to illustrate how the normalized conductance bounds the rate of convergence. In each case we compute the mixing time for the uniform probability function on the vertices. Our first examples will be simple graphs. The graphs do not have rapid converge, but their simplicity helps illustrate how to bound the normalized conductance and hence the rate of convergence.

A 1-dimensional lattice

Consider a random walk on an undirected graph consisting of an n-vertex path with self-loops at the both ends. With the self loops, we have $p_{xy} = 1/2$ on all edges (x,y), and so the stationary distribution is a uniform $\frac{1}{n}$ over all vertices by Lemma 4.3. The set with minimum normalized conductance is the set S with probability $\pi(S) \le \frac{1}{2}$ having the smallest ratio of probability mass exiting it, $\sum_{(x,y)\in(S,\bar{S})} \pi_x p_{xy}$, to probability mass inside it, $\pi(S)$. This set consists of the first $n/2$ vertices, for which the numerator is $\frac{1}{2n}$ and denominator is $\frac{1}{2}$. Thus,

$$\Phi(S) = \frac{1}{n.}$$

By Theorem 4.5, for ε a constant such as 1/100, after $O(n^2 \log n/\epsilon^3)$ steps, $||\mathbf{a_t} - \pi||_1 \le 1/100$. This graph does not have rapid convergence. The hitting time and the cover time are $O(n^2)$. In many interesting cases, the mixing time may be much smaller than the cover time. We will see such an example later.

A 2-dimensional lattice

Consider the $n \times n$ lattice in the plane where from each point there is a transition to each of the coordinate neighbors with probability $1/4$. At the boundary there are self-loops with probability 1-(number of neighbors)/4. It is easy to see that the chain is connected.

Since $p_{ij} = p_{ji}$, the function $f_i = 1/n^2$ satisfies $f_i p_{ij} = f_j p_{ji}$ and by Lemma 4.3, **f** is the stationary distribution. Consider any subset S consisting of at most half the states. If $|S| \geq \frac{n^2}{4}$, then the subset with the fewest edges leaving it consists of some number of columns plus perhaps one additional partial column. The number of edges leaving S is at least n. Thus

$$\sum_{i \in S} \sum_{j \in \bar{S}} \pi_i p_{ij} \geq \Omega\left(n \frac{1}{n^2}\right) = \Omega\left(\frac{1}{n}\right).$$

Since $|S| \geq \frac{n^2}{4}$, in this case

$$\Phi(S) \geq \Omega\left(\frac{1/n}{\min\left(\frac{S}{n^2}, \frac{\bar{S}}{n^2}\right)}\right) = \Omega\left(\frac{1}{n}\right).$$

If $|S| < \frac{n^2}{4}$, the subset S of a given size that has the minimum number of edges leaving consists of a square located at the lower left hand corner of the grid (Exercise 4.21). If

$|S|$ is not a perfect square then the right most column of S is short. Thus at least $2\sqrt{|S|}$ points in S are adjacent to points in $\bar{S}$. Each of these points contributes $\pi_i p_{ij} = \Omega(\frac{1}{n^2})$ to the flow($S,\bar{S}$). Thus,

$$\sum_{i \in S} \sum_{j \in \bar{S}} \pi_i p_{ij} \geq \frac{c\sqrt{|S|}}{n^2}$$

and

$$\Phi(S) = \frac{\sum_{i \in S} \sum_{j \in \bar{S}} \pi_i p_{ij}}{\min\left(\pi(S), \pi(\bar{S})\right)} \geq \frac{c\sqrt{|S|}/n^2}{|S|/n^2} = \frac{c}{\sqrt{|S|}} = \Omega\left(\frac{1}{n}\right).$$

Thus, in either case, after $O(n^2 \ln n/\epsilon^3)$ steps, $|\mathbf{a}(\mathbf{t}) - \boldsymbol{\pi}|_1 \leq \epsilon$.

A lattice in d-dimensions

Next consider the $n \times n \times \cdots \times n$ lattice in d-dimensions with a self-loop at each boundary point with probability 1 – (number of neighbors)/$2d$. The self loops make all π_i equal to n^{-d}. View the lattice as an undirected graph and consider the random walk on this undirected graph. Since there are n^d states, the cover time is at least n^d and thus exponentially dependent on d. It is possible to show (Exercise 4.22) that Φ is $\Omega(\frac{1}{dn})$. Since all π_i are equal to n^{-d}, the mixing time is $O(d^3 n^2 \ln n/\varepsilon^3)$, which is polynomially bounded in n and d.

The d-dimensional lattice is related to the Metropolis-Hastings algorithm and Gibbs sampling although in those constructions there is a nonuniform probability distribution at the vertices. However, the d-dimension lattice case suggests why the Metropolis-Hastings and Gibbs sampling constructions might converge fast.

A clique

Consider an n vertex clique with a self loop at each vertex. For each edge, $p_{xy} = \frac{1}{n}$ and thus for each vertex, $\pi_x = \frac{1}{n}$. Let S be a subset of the vertices. Then

$$\sum_{x \in S} \pi_x = \frac{|S|}{n}.$$

$$\sum_{(x,y) \in (S,\bar{S})} \pi_x p_{xy} = \pi_x p_{xy} |S||\bar{S}| = \frac{1}{n^2} |S||\bar{S}|$$

and

$$\Phi(S) = \frac{\sum_{(x,y)\in(S,\bar{S})} \pi_x p_{xy}}{\min(\sum_{x\in S} \pi_x, \sum_{x\in\bar{S}} \pi_x)} = \frac{\frac{1}{n^2}|S||\overline{S}|}{\min(\frac{1}{n}|S|, \frac{1}{n}|\overline{S}|)} = \frac{1}{n}\max(|S|, |\overline{S}|) = \frac{1}{2}.$$

This gives a bound on the ε-mixing time of

$$O\left(\frac{\ln \frac{1}{\pi_{\min}}}{\Phi^2\epsilon^3}\right) = O\left(\frac{\ln n}{\epsilon^3}\right)$$

95 .

However, a walker on the clique starting from any probability distribution will in one step be exactly at the stationary probability distribution.

A connected undirected graph

Next consider a random walk on a connected n vertex undirected graph where at each vertex all edges are equally likely. The stationary probability of a vertex equals the degree of the vertex divided by the sum of degrees. That is, if the degree of vertex x is d_x and the number of edges in the graph is m, then $\pi_x = \frac{d_x}{2m}$. Notice that for any edge (x,y) we have

$$\pi_x p_{xy} = \left(\frac{d_x}{2m}\right)\left(\frac{1}{d_x}\right) = \frac{1}{2m}.$$

Therefore, for any S, the total conductance of edges out of S is at least $\frac{1}{2m}$, and so Φ is at least $\frac{1}{m}$. Since $\pi_{\min} \geq \frac{1}{2m} \geq \frac{1}{n^2}$, $\ln\frac{1}{\pi_{min}} = O(\ln n)$. Thus, the mixing time is $O(m^2 \ln n/\varepsilon^3) = O(n^4 \ln n/\varepsilon^3)$.

The Gaussian distribution on the interval [-1,1]

Consider the interval $[-1,1]$. Let δ be a "grid size" specified later and let G be the graph consisting of a path on the $\frac{2}{\delta}+1$ vertices $\{-1,-1+\delta,-1+2\delta,\ldots,1-\delta,1\}$ having self loops at the two ends. Let $\pi_x = ce^{-\alpha x^2}$ for $x \in \{-1,-1+\delta,-1+2\delta,\ldots,1-\delta,1\}$ where $\alpha > 1$ and c has been adjusted so that $\sum_x \pi_x = 1$.

We now describe a simple Markov chain with the π_x as its stationary probability and argue its fast convergence. With the Metropolis-Hastings' construction, the transition

$$p_{x,x+\delta} = \frac{1}{2}\min\left(1, \frac{e^{-\alpha(x+\delta)^2}}{e^{-\alpha x^2}}\right) \qquad p_{x,x-\delta} = \frac{1}{2}\min\left(1, \frac{e^{-\alpha(x-\delta)^2}}{e^{-\alpha x^2}}\right)$$

probabilities are

! and.

Let S be any subset of states with $\boldsymbol{\pi}(S) \leq \frac{1}{2}$. First consider the case when S is an interval $[k\delta,1]$ for $k \geq 2$. It is easy to see that

$$\begin{aligned}\boldsymbol{\pi}(S) &\leq \int_{x=(k-1)\delta}^{\infty} ce^{-\alpha x^2}\, dx \\ &\leq \int_{(k-1)\delta}^{\infty} \frac{x}{(k-1)\delta} ce^{-\alpha x^2}\, dx \\ &= O\left(\frac{ce^{-\alpha((k-1)\delta)^2}}{\alpha(k-1)\delta}\right).\end{aligned}$$

Now there is only one edge from S to $\bar{S}$ and total conductance of edges out of S is

$$\sum_{i\in S}\sum_{j\notin S} \pi_i p_{ij} = \pi_{k\delta} p_{k\delta,(k-1)\delta} = \min(ce^{-\alpha k^2\delta^2}, ce^{-\alpha(k-1)^2\delta^2}) = ce^{-\alpha k^2\delta^2}.$$

Using $2 \leq k \leq 1/\delta$, $\alpha \geq 1$, and $\pi(\bar{S}) \leq 1$,

$$\Phi(S) = \frac{\text{flow}(S,\bar{S})}{\min(\pi(S),\pi(\bar{S}))} \geq \frac{ce^{-\alpha k^2\delta^2}}{\Omega(\alpha(k-1)\delta e^{-\alpha\delta^2(2k-1)})} \geq \frac{ce^{-\alpha((k-1)\delta)^2}}{\Omega(\alpha\delta e^{-O(\alpha\delta)})}.$$

For the grid size less than the variance of the Gaussian distribution, $\delta < \frac{1}{\alpha}$, we have $\alpha\delta < 1$, so $e^{-O(\alpha\delta)} = \Omega(1)$, thus, $\Phi(S) \geq \Omega(\alpha\delta)$. Now, $\pi_{\min} \geq ce^{-\alpha} \geq e^{-1/\delta}$, so $\ln(1/\pi_{\min}) \leq 1/\delta$.

If S is not an interval of the form $[k,1]$ or $[-1,k]$, then the situation is only better since there is more than one "boundary" point which contributes to flow$(S,\bar{S})$. We do not present this argument here. By Theorem 4.5 in $\Omega(1/\alpha^2\delta^3\varepsilon^3)$ steps, a walk gets within ε of the steady state distribution.

In the uniform probability case the -mixing time is bounded by $n^2 \log n$. For comparison, in the Gaussian case set $\delta = 1/n$ and $\alpha = 1/3$. This gives an -mixing time bound of n^3. In the Gaussian case with the entire initial probability on the first vertex, the chain begins to converge faster to the stationary probability than the uniform distribution case since the chain favors higher degree vertices. However, ultimately the distribution must reach the lower probability vertices on the other side of the Gaussian's maximum and here the chain is slower since it favors not leaving the higher probability vertices.

In these examples, we have chosen simple probability distributions. The methods extend to more complex situations.

4.5 Electrical Networks and Random Walks

In the next few sections, we study the relationship between electrical networks and random walks on undirected graphs. The graphs have nonnegative weights on each edge. A step is executed by picking a random edge from the current vertex with probability proportional to the edge's weight and traversing the edge.

An electrical network is a connected, undirected graph in which each edge (x,y) has a resistance $r_{xy} > 0$. In what follows, it is easier to deal with conductance defined as the reciprocal of resistance, $c_{xy} = \frac{1}{r_{xy}}$, rather than resistance. Associated with an electrical network is a random walk on the underlying graph defined by assigning a probability $p_{xy} = c_{xy}/c_x$ to the edge (x,y) incident to the vertex x, where the normalizing constant c_x equals $\sum_y c_{xy}$. Note that although c_{xy} equals c_{yx}, the probabilities p_{xy} and p_{yx} may not be equal due to the normalization required to make the probabilities at each vertex sum to one. We shall soon see that there is a relationship between current flowing in an electrical network and a random walk on the underlying graph.

Since we assume that the undirected graph is connected, by Theorem 4.2 there is a unique stationary probability distribution.The stationary probability distribution is π where $\pi_x = \frac{c_x}{c_0}$ with $c_0 = \sum_x c_x$. To see this, for all x and y

$$\pi_x p_{xy} = \frac{c_x}{c_0}\frac{c_{xy}}{c_x} = \frac{c_y}{c_0}\frac{c_{yx}}{c_y} = \pi_y p_{yx}$$

and hence by Lemma 4.3, π is the unique stationary probability.

Harmonic functions

Harmonic functions are useful in developing the relationship between electrical networks and random walks on undirected graphs. Given an undirected graph, designate a nonempty set of vertices as boundary vertices and the remaining vertices as interior vertices. A harmonic function g on the vertices is a function whose value at the boundary vertices is fixed to some boundary condition, and whose value at any interior vertex x is a weighted average of its values at all the adjacent vertices y, with weights p_{xy} satisfying $\sum_y p_{xy} = 1$ for each x. Thus, if at every interior vertex x for some set of weights p_{xy} satisfying $\sum_y p_{xy} = 1$, $g_x = \sum_y g_y p_{xy}$, then g is an harmonic function.

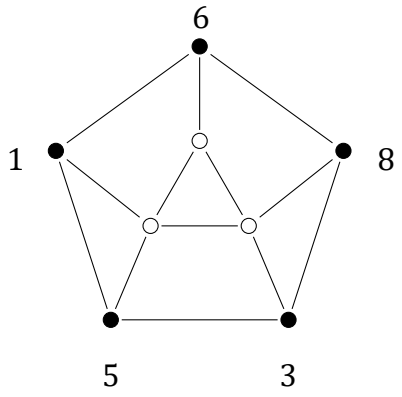
Example: Convert an electrical network with conductances c_{xy} to a weighted, undirected graph with probabilities p_{xy}. Let **f** be a function satisfying **f***P* = **f** where *P* is the matrix of probabilities. It follows that the function $g_x = \frac{f_x}{c_x}$ is harmonic.

$$g_x = \frac{f_x}{c_x} = \frac{1}{c_x}\sum_y f_y p_{yx} = \frac{1}{c_x}\sum_y f_y \frac{c_{yx}}{c_y}$$
$$= \frac{1}{c_x}\sum_y f_y \frac{c_{xy}}{c_y} = \sum_y \frac{f_y}{c_y}\frac{c_{xy}}{c_x} = \sum_y g_y p_{xy}$$

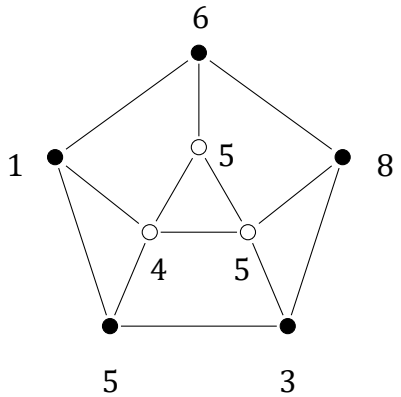
■

A harmonic function on a connected graph takes on its maximum and minimum on the boundary. This is easy to see for the following reason. Suppose the maximum does not occur on the boundary. Let *S* be the set of vertices at which the maximum value is attained. Since *S* contains no boundary vertices, $\bar{S}$ is nonempty. Connectedness implies that there is at least one edge (*x*,*y*) with $x \in S$ and $y \in \bar{S}$. The value of the function at *x* is the weighted average of the value at its neighbors, all of which are less than or equal to the value at *x* and the value at *y* is strictly less, a contradiction. The proof for the minimum value is identical.

There is at most one harmonic function satisfying a given set of equations and boundary conditions. For suppose there were two solutions, $f(x)$ and $g(x)$. The difference of two



Graph with boundary vertices dark and boundary conditions specified.



Values of harmonic function satisfying boundary conditions where the edge weights at each vertex are equal

Figure 4.7: Graph illustrating an harmonic function.

solutions is itself harmonic. Since $h(x) = f(x) - g(x)$ is harmonic and has value zero on the boundary, by the min and max principles it has value zero everywhere. Thus $f(x) = g(x)$.

The analogy between electrical networks and random walks

There are important connections between electrical networks and random walks on undirected graphs. Choose two vertices a and b. Attach a voltage source between a and b so that the voltage v_a equals one volt and the voltage v_b equals zero. Fixing the voltages at v_a and v_b induces voltages at all other vertices, along with a current flow through the edges of the network. What we will show below is the following. Having fixed the voltages at the vertices a and b, the voltage at an arbitrary vertex x equals the probability that a random walk that starts at x will reach a before it reaches b. We will also show there is a related probabilistic interpretation of current as well.

Probabilistic interpretation of voltages

Before relating voltages and probabilities, we first show that the voltages form a harmonic function. Let x and y be adjacent vertices and let i_{xy} be the current flowing through the edge from x to y. By Ohm's law,

$$i_{xy} = \frac{v_x - v_y}{r_{xy}} = (v_x - v_y)c_{xy}$$

.

By Kirchhoff's law the currents flowing out of each vertex sum to zero.

$$\sum_y i_{xy} = 0$$

Replacing currents in the above sum by the voltage difference times the conductance yields

$$\sum_y (v_x - v_y)c_{xy} = 0$$

or $v_x \sum_y c_{xy} = \sum_y v_y c_{xy}$.

Observing that $\sum_y c_{xy} = c_x$ and that $p_{xy} = \frac{c_{xy}}{c_x}$, yields $v_x c_x = \sum_y v_y p_{xy} c_x$. Hence, $v_x = \sum_y v_y p_{xy}$. Thus, the voltage at each vertex x is a weighted average of the voltages at the adjacent vertices. Hence the voltages form a harmonic function with $\{a,b\}$ as the boundary.

Let p_x be the probability that a random walk starting at vertex x reaches a before b.

Clearly $p_a = 1$ and $p_b = 0$. Since $v_a = 1$ and $v_b = 0$, it follows that $p_a = v_a$ and $p_b = v_b$. Furthermore, the probability of the walk reaching a from x before reaching b is the sum over all y adjacent to x of the probability of the walk going from x to y in the first step and then reaching a from y before reaching b. That is

$$p_x = \sum_y p_{xy} p_y.$$

Hence, p_x is the same harmonic function as the voltage function v_x and **v** and **p** satisfy the same boundary conditions at a and b.. Thus, they are identical functions. The probability of a walk starting at x reaching a before reaching b is the voltage v_x.

Probabilistic interpretation of current

In a moment, we will set the current into the network at a to have a value which we will equate with one random walk. We will then show that the current i_{xy} is the net frequency with which a random walk from a to b goes through the edge xy before reaching b. Let u_x be the expected number of visits to vertex x on a walk from a to b before reaching b. Clearly $u_b = 0$. Consider a node x not equal to a or b. Every time the walk visits x, it must have come from some neighbor y. Thus, the expected number of visits to x before reaching b is the sum over all neighbors y of the expected number of visits u_y to y before reaching b times the probability p_{yx} of going from y to x. That is,

$$u_x = \sum_y u_y p_{yx}.$$

Since $c_x p_{xy} = c_y p_{yx}$

$$u_x = \sum_y u_y \frac{c_x p_{xy}}{c_y}$$

and hence $\frac{u_x}{c_x} = \sum_y \frac{u_y}{c_y} p_{xy}$. It follows that $\frac{u_x}{c_x}$ is harmonic with a and b as the boundary where the boundary conditions are $u_b = 0$ and u_a equals some fixed value. Now, $\frac{u_b}{c_b} = 0.$ Setting the current into a to one, fixed the value of v_a. Adjust the current into a so that v_a equals $\frac{u_a}{c_a}$. Now $\frac{u_x}{c_x}$ and v_x satisfy the same boundary conditions and thus are the same harmonic function. Let the current into a correspond to one walk. Note that if the walk starts at a and ends at b, the expected value of the difference between the number of times the walk leaves a and enters a must be one. This implies that the amount of current into a corresponds to one walk.

Next we need to show that the current i_{xy} is the net frequency with which a random walk traverses edge xy.

$$i_{xy} = (v_x - v_y)c_{xy} = \left(\frac{u_x}{c_x} - \frac{u_y}{c_y}\right) c_{xy} = u_x \frac{c_{xy}}{c_x} - u_y \frac{c_{xy}}{c_y} = u_x p_{xy} - u_y p_{yx}$$

The quantity $u_x p_{xy}$ is the expected number of times the edge xy is traversed from x to y and the quantity $u_y p_{yx}$ is the expected number of times the edge xy is traversed from y to x. Thus, the current i_{xy} is the expected net number of traversals of the edge xy from x to y.

Effective resistance and escape probability

Set $v_a = 1$ and $v_b = 0$. Let i_a be the current flowing into the network at vertex a and out at vertex b. Define the *effective resistance* r_{eff} between a and b to be $r_{eff} = \frac{v_a}{i_a}$ and the *effective conductance* c_{eff} to be $c_{eff} = \frac{1}{r_{eff}}$. Define the *escape probability*, p_{escape}, to be the probability that a random walk starting at a reaches b before returning to a. We now show that the escape probability is $\frac{c_{eff}}{c_a}$. For convenience, assume that a and b are not adjacent. A slight modification of the argument suffices for the case when a and b are adjacent.

$$i_a = \sum_y (v_a - v_y)c_{ay}$$

Since $v_a = 1$,

$$i_a = \sum_y c_{ay} - c_a \sum_y v_y \frac{c_{ay}}{c_a}$$
$$= c_a \left[1 - \sum_y p_{ay} v_y\right].$$

For each y adjacent to the vertex a, p_{ay} is the probability of the walk going from vertex a to vertex y. Earlier we showed that v_y is the probability of a walk starting at y going to a before reaching b. Thus, $\sum_y p_{ay} v_y$ is the probability of a walk starting at a returning to a before reaching b and $1 - \sum_y p_{ay} v_y$ is the probability of a walk starting at a reaching b before returning to a. Thus, $i_a = c_a p_{escape}$. Since $v_a = 1$ and $c_{eff} = \frac{i_a}{v_a}$, it follows that $c_{eff} = i_a$. Thus, $c_{eff} = c_a p_{escape}$ and hence $p_{escape} = \frac{c_{eff}}{c_a}$.

For a finite connected graph, the escape probability will always be nonzero. Consider an infinite graph such as a lattice and a random walk starting at some vertex a. Form a series of finite graphs by merging all vertices at distance d or greater from a into a single vertex b for larger and larger values of d. The limit of p_{escape} as d goes to infinity is the probability that the random walk will never return to a. If $p_{escape} \to 0$, then eventually any

random walk will return to a. If $p_{escape} \rightarrow q$ where $q > 0$, then a fraction of the walks never return. Thus, the escape probability terminology.

4.6 Random Walks on Undirected Graphs with Unit Edge Weights

We now focus our discussion on random walks on undirected graphs with uniform edge weights. At each vertex, the random walk is equally likely to take any edge. This corresponds to an electrical network in which all edge resistances are one. Assume the graph is connected. We consider questions such as what is the expected time for a random walk starting at a vertex x to reach a target vertex y, what is the expected time until the random walk returns to the vertex it started at, and what is the expected time to reach every vertex?

Hitting time

The *hitting time* h_{xy}, sometimes called *discovery time*, is the expected time of a random walk starting at vertex x to reach vertex y. Sometimes a more general definition is given where the hitting time is the expected time to reach a vertex y from a given starting probability distribution.

One interesting fact is that adding edges to a graph may either increase or decrease h_{xy} depending on the particular situation. Adding an edge can shorten the distance from x to y thereby decreasing h_{xy} or the edge could increase the probability of a random walk going to some far off portion of the graph thereby increasing h_{xy}. Another interesting fact is that hitting time is not symmetric. The expected time to reach a vertex y from a vertex x in an undirected graph may be radically different from the time to reach x from y.

We start with two technical lemmas. The first lemma states that the expected time to traverse a path of n vertices is $\Theta(n^2)$.

Lemma 4.7 *The expected time for a random walk starting at one end of a path of n vertices to reach the other end is* $\Theta(n^2)$.

Proof: Consider walking from vertex 1 to vertex n in a graph consisting of a single path of n vertices. Let h_{ij}, $i < j$, be the hitting time of reaching j starting from i. Now $h_{12} = 1$ and

$$h_{i,i+1} = \tfrac{1}{2} + \tfrac{1}{2}(1 + h_{i-1,i+1}) = 1 + \tfrac{1}{2}(h_{i-1,i} + h_{i,i+1}) \quad 2 \leq i \leq n-1.$$

Solving for $h_{i,i+1}$ yields the recurrence

$$h_{i,i+1} = 2 + h_{i-1,i}.$$

Solving the recurrence yields $h_{i,i+1} = 2i - 1$.

To get from 1 to n, you need to first reach 2, then from 2 (eventually) reach 3, then from 3 (eventually) reach 4, and so on. Thus by linearity of expectation,

$$\begin{aligned} h_{1,n} &= \sum_{i=1}^{n-1} h_{i,i+1} = \sum_{i=1}^{n-1} (2i-1) \\ &= 2\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} 1 \\ &= 2\frac{n(n-1)}{2} - (n-1) \\ &= (n-1)^2 . \end{aligned}$$

■

The next lemma shows that the expected time spent at vertex i by a random walk from vertex 1 to vertex n in a chain of n vertices is $2(i-1)$ for $2 \le i \le n-1$.

Lemma 4.8 *Consider a random walk from vertex 1 to vertex n in a chain of n vertices. Let t(i) be the expected time spent at vertex i. Then*

$$t(i) = \begin{cases} n-1 & i = 1 \\ 2(n-i) & 2 \le i \le n-1 \\ 1 & i = n. \end{cases}$$

Proof: Now $t(n) = 1$ since the walk stops when it reaches vertex n. Half of the time when the walk is at vertex $n-1$ it goes to vertex n. Thus $t(n-1) = 2$. For $3 \le i < n-1$, $t(i) = \frac{1}{2}[t(i-1) + t(i+1)]$ and $t(1)$ and $t(2)$ satisfy $t(1) = \frac{1}{2}t(2) + 1$ and $t(2) = t(1) + \frac{1}{2}t(3)$. Solving for $t(i+1)$ for $3 \le i < n-1$ yields

$$t(i+1) = 2t(i) - t(i-1)$$

which has solution $t(i) = 2(n-i)$ for $3 \le i < n-1$. Then solving for $t(2)$ and $t(1)$ yields $t(2) = 2(n-2)$ and $t(1) = n-1$. Thus, the total time spent at vertices is

$$n - 1 + 2(1 + 2 + \cdots + n - 2) + 1 = (n-1) + 2\frac{(n-1)(n-2)}{2} + 1 = (n-1)^2 + 1$$

which is one more than h_{1n} and thus is correct. ■

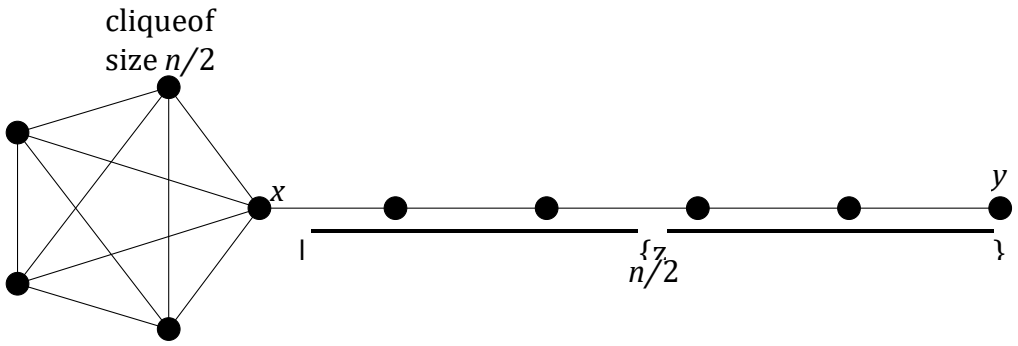


Figure 4.8: Illustration that adding edges to a graph can either increase or decrease hitting time.

Adding edges to a graph might either increase or decrease the hitting time h_{xy}. Consider the graph consisting of a single path of n vertices. Add edges to this graph to get the graph in Figure 4.8 consisting of a clique of size $n/2$ connected to a path of $n/2$ vertices. Then add still more edges to get a clique of size n. Let x be the vertex at the midpoint of the original path and let y be the other endpoint of the path consisting of $n/2$ vertices as shown in the figure. In the first graph consisting of a single path of length n, $h_{xy} = \Theta(n^2)$. In the second graph consisting of a clique of size $n/2$ along with a path of length $n/2$, $h_{xy} = \Theta(n^3)$.To see this latter statement, note that starting at x, the walk will go down the path towards y and return to x for $n/2-1$ times on average before reaching y for the first time, by Lemma 4.8. Each time the walk in the path returns to x, with probability $(n/2 - 1)/(n/2)$ it enters the clique and thus on average enters the clique $\Theta(n)$ times before starting down the path again. Each time it enters the clique, it spends $\Theta(n)$ time in the clique before returning to x. It then reenters the clique $\Theta(n)$ times before starting down the path to y. Thus, each time the walk returns to x from the path it spends $\Theta(n^2)$ time in the clique before starting down the path towards y for a total expected time that is $\Theta(n^3)$ before reaching y. In the third graph, which is the clique of size n, $h_{xy} = \Theta(n)$. Thus, adding edges first increased h_{xy} from n^2 to n^3 and then decreased it to n.

Hitting time is not symmetric even in the case of undirected graphs. In the graph of Figure 4.8, the expected time, h_{xy}, of a random walk from x to y, where x is the vertex of attachment and y is the other end vertex of the chain, is $\Theta(n^3)$. However, h_{yx} is $\Theta(n^2)$.

Commute time

The *commute time*, commute(x,y), is the expected time of a random walk starting at x reaching y and then returning to x. So commute(x,y) = $h_{xy} + h_{yx}$. Think of going from home to office and returning home. Note that commute time is symmetric. We now relate the commute time to an electrical quantity, the effective resistance. The *effective resistance*

between two vertices x and y in an electrical network is the voltage difference between x and y when one unit of current is inserted at vertex x and withdrawn from vertex y.

Theorem 4.9 *Given a connected, undirected graph, consider the electrical network where each edge of the graph is replaced by a one ohm resistor. Given vertices x and y, the commute time, commute*(*x,y*)*, equals* $2mr_{xy}$ *where* r_{xy} *is the effective resistance from x to y and m is the number of edges in the graph.*

Proof: Insert at each vertex i a current equal to the degree d_i of vertex i. The total current inserted is $2m$ where m is the number of edges. Extract from a specific vertex j all of this $2m$ current (note: for this to be legal, the graph must be connected). Let v_{ij} be the voltage difference from i to j. The current into i divides into the d_i resistors at vertex i. The current in each resistor is proportional to the voltage across it. Let k be a vertex adjacent to i. Then the current through the resistor between i and k is $v_{ij} - v_{kj}$, the voltage drop across the resistor. The sum of the currents out of i through the resistors must equal d_i, the current injected into i.

$$d_i = \underset{k \text{ adj}}{\text{X}} (v_{ij} - v_{kj}) = d_i v_{ij} - \underset{k \text{ adj to } i}{\text{X}} v_{kj}.$$

Solving for v_{ij}

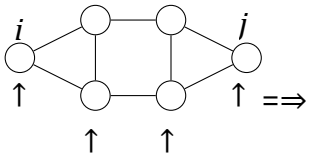
$$v_{ij} = 1 + \sum_{\substack{k \text{ adj} \\ \text{to } i}} \frac{1}{d_i} v_{kj} = \sum_{\substack{k \text{ adj} \\ \text{to } i}} \frac{1}{d_i}(1 + v_{kj}) \quad . \tag{4.11}$$

Now the hitting time from i to j is the average time over all paths from i to k adjacent to i and then on from k to j. This is given by

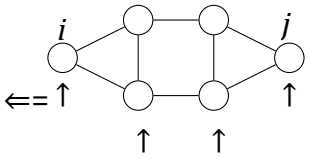
$$h_{ij} = \sum_{\substack{k \text{ adj} \\ \text{to } i}} \frac{1}{d_i}(1 + h_{kj}) \quad . \tag{4.12}$$

Subtracting (4.12) from (4.11), gives $v_{ij} - h_{ij} = \sum_{\substack{k \text{ adj} \\ \text{to } i}} \frac{1}{d_i}(v_{kj} - h_{kj})$. Thus, the function $v_{ij} - h_{ij}$ is harmonic. Designate vertex j as the only boundary vertex. The value of $v_{ij} - h_{ij}$ at $i = j$, namely $v_{jj} - h_{jj}$, is zero, since both v_{jj} and h_{jj} are zero. So the function $v_{ij} - h_{ij}$ must be zero everywhere. Thus, the voltage v_{ij} equals the expected time h_{ij} from i to j.

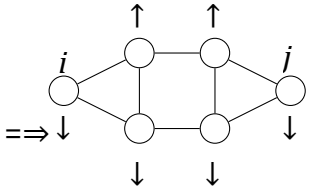
To complete the proof of Theorem 4.9, note that $h_{ij} = v_{ij}$ is the voltage from i to j when currents are inserted at all vertices in the graph and extracted at vertex j. If the current is extracted from i instead of j, then the voltages change and $v_{ji} = h_{ji}$ in the new ↓ ↓ ↓ ↓



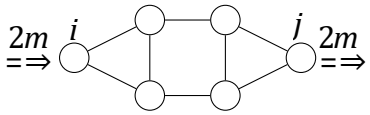
Insert current at each vertex equal to degree of the vertex.
Extract $2m$ at vertex j, $v_{ij} = h_{ij}$.
(a)



Extract current from i instead of j.
For new voltages $v_{ji} = h_{ji}$.
(b)



Reverse currents in (b).
For new voltages $-v_{ji} = h_{ji}$.
Since $-v_{ji} = v_{ij}, h_{ji} = v_{ij}$.
(c)



Superpose currents in (a) and (c).
$2mr_{ij} = v_{ij} = h_{ij} + h_{ji} = commute(i,j)$.
(d)

Figure 4.9: Illustration of proof that commute(x,y) = $2mr_{xy}$ where m is the number of edges in the undirected graph and r_{xy} is the effective resistance between x and y.

setup. Finally, reverse all currents in this latter step. The voltages change again and for the new voltages $-v_{ji} = h_{ji}$. Since $-v_{ji} = v_{ij}$, we get $h_{ji} = v_{ij}$.

Thus, when a current is inserted at each vertex equal to the degree of the vertex and the current is extracted from j, the voltage v_{ij} in this set up equals h_{ij}. When we extract the current from i instead of j and then reverse all currents, the voltage v_{ij} in this new set up equals h_{ji}. Now, superpose both situations, i.e., add all the currents and voltages. By linearity, for the resulting v_{ij}, which is the sum of the other two v_{ij}'s, is $v_{ij} = h_{ij} + h_{ji}$. All currents into or out of the network cancel except the $2m$ amps injected at i and withdrawn at j. Thus, $2mr_{ij} = v_{ij} = h_{ij} + h_{ji}$ = commute(i,j) or commute(i,j) = $2mr_{ij}$ where r_{ij} is the effective resistance from i to j.■

The following corollary follows from Theorem 4.9 since the effective resistance r_{uv} is less than or equal to one when u and v are connected by an edge.

Corollary 4.10 *If vertices x and y are connected by an edge, then* $h_{xy} + h_{yx} \leq 2m$ *where m is the number of edges in the graph.*

Proof: If x and y are connected by an edge, then the effective resistance r_{xy} is less than or equal to one. ■

Corollary 4.11 *For vertices x and y in an n vertex graph, the commute time, commute*(x,y)*, is less than or equal to* n^3.

Proof: By Theorem 4.9 the commute time is given by the formula commute$(x,y) = 2mr_{xy}$ where m is the number of edges. In an n vertex graph there exists a path from x to y of length at most n. Since the resistance can not be greater than that of any path from x to y, $r_{xy} \leq n$. Since the number of edges is at most $\binom{n}{2}$

$$\text{commute}(x, y) = 2mr_{xy} \leq 2\binom{n}{2} n \cong n^3 .$$

■

While adding edges into a graph can never increase the effective resistance between two given nodes x and y, it may increase or decrease the commute time. To see this consider three graphs: the graph consisting of a chain of n vertices, the graph of Figure 4.8, and the clique on n vertices.

Cover time

The *cover time*, cover(x,G), is the expected time of a random walk starting at vertex x in the graph G to reach each vertex at least once. We write cover(x) when G is understood. The cover time of an undirected graph G, denoted cover(G), is

$$\text{cover}(G) = \max_x \text{cover}(x,G).$$

For cover time of an undirected graph, increasing the number of edges in the graph may increase or decrease the cover time depending on the situation. Again consider three graphs, a chain of length n which has cover time $\Theta(n^2)$, the graph in Figure 4.8 which has cover time $\Theta(n^3)$, and the complete graph on n vertices which has cover time $\Theta(n\log n)$. Adding edges to the chain of length n to create the graph in Figure 4.8 increases the cover time from n^2 to n^3 and then adding even more edges to obtain the complete graph reduces the cover time to $n\log n$.

Note: The cover time of a clique is $\Theta(n\log n)$ since this is the time to select every integer out of n integers with high probability, drawing integers at random. This is called the *coupon collector problem*. The cover time for a straight line is $\Theta(n^2)$ since it is the same as

the hitting time. For the graph in Figure 4.8, the cover time is $\Theta(n^3)$ since one takes the maximum over all start states and cover$(x,G) = \Theta(n^3)$ where x is the vertex of attachment.

Theorem 4.12 *Let G be a connected graph with n vertices and m edges. The time for a random walk to cover all vertices of the graph G is bounded above by* $4m(n-1)$.
Proof: Consider a depth first search of the graph G starting from some vertex z and let T be the resulting depth first search spanning tree of G. The depth first search covers every vertex. Consider the expected time to cover every vertex in the order visited by the depth first search. Clearly this bounds the cover time of G starting from vertex z. Note that each edge in T is traversed twice, once in each direction.

$$\text{cover}(z,G) \leq \sum_{\substack{(x,y)\in T \\ (y,x)\in T}} h_{xy}.$$

If (x,y) is an edge in T, then x and y are adjacent and thus Corollary 4.10 implies $h_{xy} \leq 2m$. Since there are $n-1$ edges in the dfs tree and each edge is traversed twice, once in each direction, cover$(z) \leq 4m(n-1)$. This holds for all starting vertices z. Thus, cover$(G) \leq 4m(n-1)$. ■

The theorem gives the correct answer of n^3 for the $n/2$ clique with the $n/2$ tail. It gives an upper bound of n^3 for the n-clique where the actual cover time is $n\log n$.

Let r_{xy} be the effective resistance from x to y. Define the resistance $r_{eff}(G)$ of a graph G by $r_{eff}(G) = \max_{x,y}(r_{xy})$.

Theorem 4.13 *Let G be an undirected graph with m edges. Then the cover time for G is bounded by the following inequality*

$$mr_{eff}(G) \leq cover(G) \leq 6emr_{eff}(G)\ln n + n$$

where $e \approx 2.718$ *is Euler's constant and* $r_{eff}(G)$ *is the resistance of G.* **Proof:** By definition $r_{eff}(G) = \max_{x,y}(r_{xy})$. Let u and v be the vertices of G for which r_{xy} is maximum. Then $r_{eff}(G) = r_{uv}$. By Theorem 4.9, commute$(u,v) = 2mr_{uv}$. Hence $mr_{uv} = \frac{1}{2}$ commute(u,v). Note that $\frac{1}{2}$commute(u,v) is the average of h_{uv} and h_{vu}, which is clearly less than or equal to $\max(h_{uv},h_{vu})$. Finally, $\max(h_{uv},h_{vu})$ is less than or equal to

max(cover(u,G),cover(v,G)) which is clearly less than the cover time of G. Putting these facts together gives the first inequality in the theorem.

$$m\, r_{eff}(G) = mr_{uv} = \tfrac{1}{2}\text{commute}(u,v) \le \max(h_{uv},h_{vu}) \le \text{cover}(G)$$

For the second inequality in the theorem, by Theorem 4.9, for any x and y, commute(x,y) equals $2mr_{xy}$ which is less than or equal to $2mr_{eff}(G)$, implying $h_{xy} \le 2mr_{eff}(G)$. By the Markov inequality, since the expected time to reach y starting at any x is less than $2mr_{eff}(G)$, the probability that y is not reached from x in $2mr_{eff}(G)e$ steps is at most $\frac{1}{e}$. Thus, the probability that a vertex y has not been reached in $6emr_{eff}(G)\log n$ steps is at most $\frac{1}{e}^{3\ln n} = \frac{1}{n^3}$ because a random walk of length $6emr_{eff}(G)\log n$ is a sequence of $3\log n$ random walks, each of length $2emr_{eff}(G)$ and each possibly starting from different vertices. Suppose after a walk of $6emr_{eff}(G)\log n$ steps, vertices $v_1,v_2,\ldots,v_l$ had not been reached. Walk until v_1 is reached, then v_2, etc. By Corollary 4.11 the expected time for each of these is n^3, but since each happens only with probability $1/n^3$, we effectively take $O(1)$ time per v_i, for a total time at most n. More precisely, cover(G) $\le 6emr_{eff}(G)\log n + \sum_v$Prob($v$ was not visited in the first $6emr_{eff}(G)$ steps)n^3

$$\le 6em\, r_{eff}(G)\log n + \sum_v \frac{1}{n^3} n^3 \le 6em\, r_{eff}(G) + n.$$

■

4.7 Random Walks in Euclidean Space

Many physical processes such as Brownian motion are modeled by random walks. Random walks in Euclidean d-space consisting of fixed length steps parallel to the coordinate axes are really random walks on a d-dimensional lattice and are a special case of random walks on graphs. In a random walk on a graph, at each time unit an edge from the current vertex is selected at random and the walk proceeds to the adjacent vertex.

Random walks on lattices

We now apply the analogy between random walks and current to lattices. Consider a random walk on a finite segment $-n,\ldots,-1,0,1,2,\ldots,n$ of a one dimensional lattice starting from the origin. Is the walk certain to return to the origin or is there some probability that it will escape, i.e., reach the boundary before returning? The probability of reaching the boundary before returning to the origin is called the escape probability. We shall be interested in this quantity as n goes to infinity.

Convert the lattice to an electrical network by replacing each edge with a one ohm resistor. Then the probability of a walk starting at the origin reaching n or $-n$ before returning to the origin is the escape probability given by

$$= \frac{c_{eff}\, p_{escape}}{c_a}$$

where c_{eff} is the effective conductance between the origin and the boundary points and c_a is the sum of the conductances at the origin. In a d-dimensional lattice, $c_a = 2d$ assuming that the resistors have value one. For the d-dimensional lattice

$$p_{escape} = \frac{1}{2d\, r_{eff}}$$

In one dimension, the electrical network is just two series connections of n one-ohm resistors connected in parallel. So as n goes to infinity, r_{eff} goes to infinity and the escape probability goes to zero as n goes to infinity. Thus, the walk in the unbounded one Number of resistors in parallel

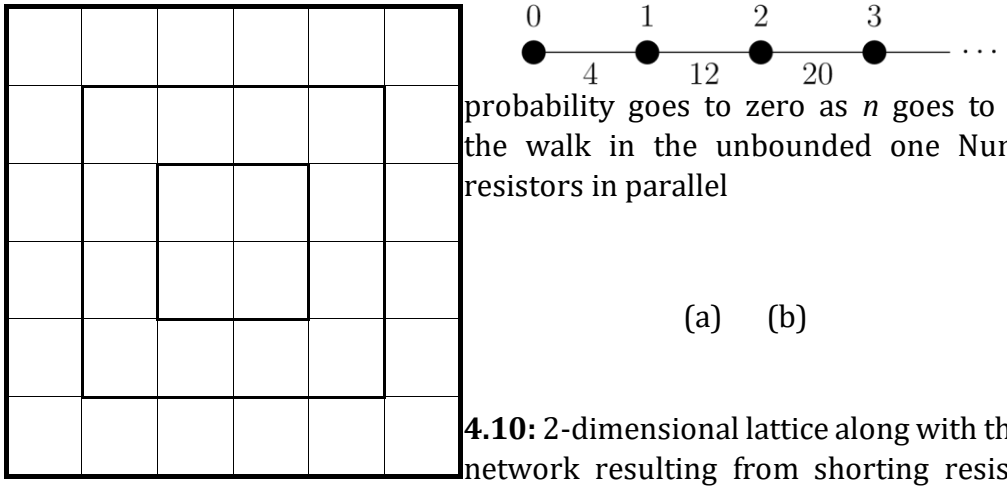


Figure 4.10: 2-dimensional lattice along with the linear network resulting from shorting resistors on the concentric squares about the origin.

dimensional lattice will return to the origin with probability one. Note, however, that the expected time to return to the origin having taken one step away, which is equal to commute(1,0), is infinite (Theorem 4.9.

Two dimensions

For the 2-dimensional lattice, consider a larger and larger square about the origin for the boundary as shown in Figure 4.10a and consider the limit of r_{eff} as the squares get larger. Shorting the resistors on each square can only reduce r_{eff}. Shorting the resistors results in the linear network shown in Figure 4.10b. As the paths get longer, the number of resistors in parallel also increases. The resistance between vertex i and $i + 1$ is really

$4(2i+1)$ unit resistors in parallel. The effective resistance of $4(2i+1)$ resistors in parallel is $1/4(2i+1)$. Thus,

$$r_{eff} \geq \tfrac{1}{4} + \tfrac{1}{12} + \tfrac{1}{20} + \cdots = \tfrac{1}{4}(1 + \tfrac{1}{3} + \tfrac{1}{5} + \cdots) = \Theta(\ln n).$$

Since the lower bound on the effective resistance and hence the effective resistance goes to infinity, the escape probability goes to zero for the 2-dimensional lattice.

Three dimensions

In three dimensions, the resistance along any path to infinity grows to infinity but the number of paths in parallel also grows to infinity. It turns out there are a sufficient number of paths that r_{eff} remains finite and thus there is a nonzero escape probability. We will prove this now. First note that shorting any edge decreases the resistance, so

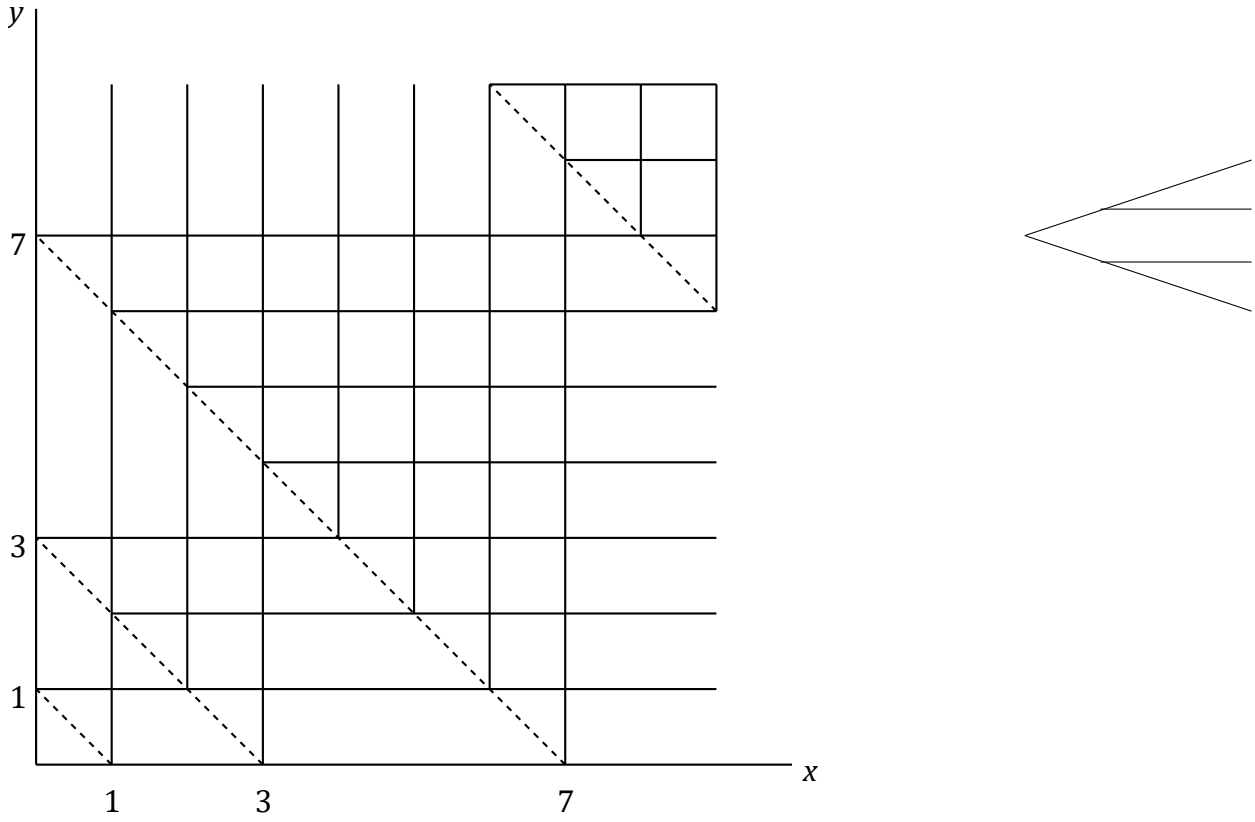


Figure 4.11: Paths in a 2-dimensional lattice obtained from the 3-dimensional construction applied in 2-dimensions.

we do not use shorting in this proof, since we seek to prove an upper bound on the resistance. Instead we remove some edges, which increases their resistance to infinity and hence increases the effective resistance, giving an upper bound. To simplify things we

consider walks on a quadrant rather than the full grid. The resistance to infinity derived from only the quadrant is an upper bound on the resistance of the full grid.

The construction used in three dimensions is easier to explain first in two dimensions, see Figure 4.11. Draw dotted diagonal lines at $x + y = 2^n - 1$. Consider two paths that start at the origin. One goes up and the other goes to the right. Each time a path encounters a dotted diagonal line, split the path into two, one which goes right and the other up. Where two paths cross, split the vertex into two, keeping the paths separate. By a symmetry argument, splitting the vertex does not change the resistance of the network. Remove all resistors except those on these paths. The resistance of the original network is less than that of the tree produced by this process since removing a resistor is equivalent to increasing its resistance to infinity.

The distances between splits increase and are 1, 2, 4, etc. At each split the number

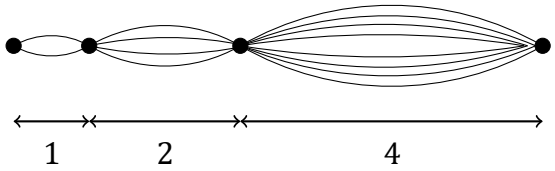


Figure 4.12: Paths obtained from 2-dimensional lattice. Distances between splits double as do the number of parallel paths.

of paths in parallel doubles. See Figure 4.12. Thus, the resistance to infinity in this two dimensional example is

$$\frac{1}{2} + \frac{1}{4}2 + \frac{1}{8}4 + \cdots = \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \cdots = \infty.$$

In the analogous three dimensional construction, paths go up, to the right, and out of the plane of the paper. The paths split three ways at planes given by $x + y + z = 2^n - 1$. Each time the paths split the number of parallel segments triple. Segments of the paths between splits are of length 1, 2, 4, etc. and the resistance of the segments are equal to the lengths. The resistance out to infinity for the tree is

$$\tfrac{1}{3} + \tfrac{1}{9}2 + \tfrac{1}{27}4 + \cdots = \tfrac{1}{3}\left(1 + \tfrac{2}{3} + \tfrac{4}{9} + \cdots\right) = \tfrac{1}{3}\frac{1}{1 - \frac{2}{3}} = 1$$

The resistance of the three dimensional lattice is less. It is important to check that the paths are edge-disjoint and so the tree is a subgraph of the lattice. Going to a subgraph is equivalent to deleting edges which increases the resistance. That is why the resistance of the lattice is less than that of the tree. Thus, in three dimensions the escape probability is nonzero. The upper bound on r_{eff} gives the lower bound

$$p_{escape} = \frac{1}{2d}\frac{1}{r_{eff}} \geq \frac{1}{6}.$$

A lower bound on r_{eff} gives an upper bound on p_{escape}. To get the upper bound on p_{escape}, short all resistors on surfaces of boxes at distances 1,2,3,, etc. Then

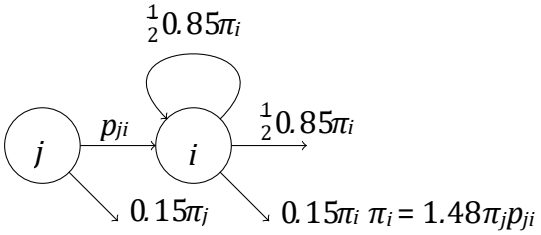
$$r_{eff} \geq \frac{1}{6}\left[1 + \frac{1}{9} + \frac{1}{25} + \cdots\right] \geq \frac{1.23}{6} \geq 0.2$$

This gives

$$p_{escape} = \frac{1}{2d}\frac{1}{r_{eff}} \leq \frac{5}{6}.$$

4.8 The Web as a Markov Chain

A modern application of random walks on directed graphs comes from trying to establish the importance of pages on the World Wide Web. Search Engines output an ordered



$$\pi_i = 0.85\pi_j p_{ji} + \frac{0.85}{2}\pi_i$$

Figure 4.13: Impact on pagerank of adding a self loop

list of webpages in response to each search query. To do this, they have to solve two problems at query time: (i) find the set of all webpages containing the query term(s) and (ii) rank the webpages and display them (or the top subset of them) in ranked order. (i) is done by maintaining a "reverse index" which we do not discuss here. (ii) cannot be done at query time since this would make the response too slow. So Search Engines rank the entire set of webpages (in the billions) "off-line" and use that single ranking for all queries. At query time, the webpages containing the query terms(s) are displayed in this ranked order.

One way to do this ranking would be to take a random walk on the web viewed as a directed graph (which we call the web graph) with an edge corresponding to each hypertext link and rank pages according to their stationary probability. Hypertext links are one-way and the web graph may not be strongly connected. Indeed, for a node at the "bottom" level there may be no out-edges. When the walk encounters this vertex the walk disappears. Another difficulty is that a vertex or a strongly connected component with no in edges is never reached. One way to resolve these difficulties is to introduce a random restart condition. At each step, with some probability *r,* jump to a vertex selected uniformly at random in the entire graph; with probability 1 − *r* select an out-edge at random from the current node and follow it. If a vertex has no out edges, the value of *r* for

that vertex is set to one. This makes the graph strongly connected so that the stationary probabilities exist.

Pagerank

The pagerank of a vertex in a directed graph is the stationary probability of the vertex, where we assume a positive restart probability of say $r = 0.15$. The restart ensures that the graph is strongly connected. The pagerank of a page is the frequency with which the page will be visited over a long period of time. If the pagerank is p, then the expected time between visits or return time is $1/p$. Notice that one can increase the pagerank of a page by reducing the return time and this can be done by creating short cycles.

Consider a vertex i with a single edge in from vertex j and a single edge out. The stationary probability π satisfies $\pi P = \pi$, and thus

$\pi_i = \pi_j p_{ji}$. Adding a

self-loop at i, results in a new equation

$$\pi_i = \pi_j p_{ji} + \frac{1}{2}\pi_i$$

or $\pi_i = 2\,\pi_j p_{ji}$.

Of course, π_j would have changed too, but ignoring this for now, pagerank is doubled by the addition of a self-loop. Adding k self loops, results in the equation

$$\pi_i = \pi_j p_{ji} + \frac{k}{k+1}\pi_i,$$

and again ignoring the change in π_j, we now have $\pi_i = (k+1)\pi_j p_{ji}$. What prevents one from increasing the pagerank of a page arbitrarily? The answer is the restart. We neglected the 0.15 probability that is taken off for the random restart. With the restart taken into account, the equation for π_i when there is no self-loop is

$$\pi_i = 0.85\pi_j p_{ji}$$

whereas, with k self-loops, the equation is

$$\pi_i = 0.85\pi_j p_{ji} + 0.85\frac{k}{k+1}\pi_i.$$

Solving for π_i yields

$$\pi_i = \frac{0.85k + 0.85}{0.15k + 1}\pi_j p_{ji}$$

which for $k = 1$ is $\pi_i = 1.48\pi_j p_{ji}$ and in the limit as $k \to \infty$ is $\pi_i = 5.67\pi_j p_{ji}$. Adding a single loop only increases pagerank by a factor of 1.74.

Relation to Hitting time

Recall the definition of hitting time h_{xy}, which for two states x and y is the expected time to reach y starting from x. Here, we deal with h_y, the average time to hit y, starting at a random node. Namely, $h_y = \frac{1}{n}\sum_x h_{xy}$, where the sum is taken over all n nodes x.
Hitting time h_y is closely related to return time and thus to the reciprocal of page rank. Return time is clearly less than the expected time until a restart plus hitting time. With r as the restart value, this gives:

$$\text{Return time to } y \leq \frac{1}{r} + h_y.$$

In the other direction, the fastest one could return would be if there were only paths of length two (assume we remove all self-loops). A path of length two would be traversed with at most probability $(1 - r)^2$. With probability $r + (1 - r)r = (2 - r)r$ one restarts and then hits v. Thus, the return time is at least $2(1 - r)^2 + (2 - r)r \times$ (hitting time). Combining these two bounds yields

$$2(1 - r)^2 + (2 - r)r(\text{hitting time}) \leq (\text{return time}) \leq \frac{1}{r} + (\text{hitting time}).$$

The relationship between return time and hitting time can be used to see if a vertex has unusually high probability of short loops. However, there is no efficient way to compute hitting time for all vertices as there is for return time. For a single vertex v, one can compute hitting time by removing the edges out of the vertex v for which one is computing hitting time and then run the pagerank algorithm for the new graph. The hitting time for v is the reciprocal of the pagerank in the graph with the edges out of v removed. Since computing hitting time for each vertex requires removal of a different set of edges, the algorithm only gives the hitting time for one vertex at a time. Since one is probably only interested in the hitting time of vertices with low hitting time, an alternative would be to use a random walk to estimate the hitting time of low hitting time vertices.

Spam

Suppose one has a web page and would like to increase its pagerank by creating other web pages with pointers to the original page. The abstract problem is the following. We are given a directed graph G and a vertex v whose pagerank we want to increase. We may add new vertices to the graph and edges from them to any vertices we want. We can also add or delete edges from v. However, we cannot add or delete edges out of other vertices.

The pagerank of v is the stationary probability for vertex v with random restarts. If we delete all existing edges out of v, create a new vertex u and edges (v,u) and (u,v), then the pagerank will be increased since any time the random walk reaches v it will be captured in the loop $v \rightarrow u \rightarrow v$. A search engine can counter this strategy by more frequent random restarts.

A second method to increase pagerank would be to create a star consisting of the vertex v at its center along with a large set of new vertices each with a directed edge to v. These new vertices will sometimes be chosen as the target of the random restart and hence the vertices increase the probability of the random walk reaching v. This second method is countered by reducing the frequency of random restarts.

Notice that the first technique of capturing the random walk increases pagerank but does not effect hitting time. One can negate the impact on pagerank of someone capturing the random walk by increasing the frequency of random restarts. The second technique of creating a star increases pagerank due to random restarts and decreases hitting time. One can check if the pagerank is high and hitting time is low in which case the pagerank is likely to have been artificially inflated by the page capturing the walk with short cycles.

Personalized pagerank

In computing pagerank, one uses a restart probability, typically 0.15, in which at each step, instead of taking a step in the graph, the walk goes to a vertex selected uniformly at random. In personalized pagerank, instead of selecting a vertex uniformly at random, one selects a vertex according to a personalized probability distribution. Often the distribution has probability one for a single vertex and whenever the walk restarts it restarts at that vertex. Note that this may make the graph disconnected.

Algorithm for computing personalized pagerank

First, consider the normal pagerank. Let α be the restart probability with which the random walk jumps to an arbitrary vertex. With probability $1 - \alpha$ the random walk selects a vertex uniformly at random from the set of adjacent vertices. Let $\mathbf{p}$ be a row vector denoting the pagerank and let A be the adjacency matrix with rows normalized to sum to one. Then $\mathbf{p} = \frac{\alpha}{n}(1,1,\ldots,1) + (1-\alpha)\,\mathbf{p}A$

$$\mathbf{p}[I - (1-\alpha)A] = \frac{\alpha}{n}(1,1,\ldots,1)$$

or

$$\mathbf{p} = \tfrac{\alpha}{n}(1,1,\ldots,1)\,[I - (1-\alpha)\,A]^{-1}.$$

Thus, in principle, $\mathbf{p}$ can be found by computing the inverse of $[I - (1 - \alpha)A]^{-1}$. But this is far from practical since for the whole web one would be dealing with matrices with

billions of rows and columns. A more practical procedure is to run the random walk and observe using the basics of the power method in Chapter 3 that the process converges to the solution **p**.

For the personalized pagerank, instead of restarting at an arbitrary vertex, the walk restarts at a designated vertex. More generally, it may restart in some specified neighborhood. Suppose the restart selects a vertex using the probability distribution *s*. Then, in the above calculation replace the vector $\frac{1}{n}(1,1,\cdots,1)$ by the vector **s**. Again, the computation could be done by a random walk. But, we wish to do the random walk calculation for personalized pagerank quickly since it is to be performed repeatedly. With more care this can be done, though we do not describe it here.

4.9 Bibliographic Notes

The material on the analogy between random walks on undirected graphs and electrical networks is from [DS84] as is the material on random walks in Euclidean space. Additional material on Markov chains can be found in [MR95b], [MU05], and [per10]. For material on Markov Chain Monte Carlo methods see [Jer98] and [Liu01].

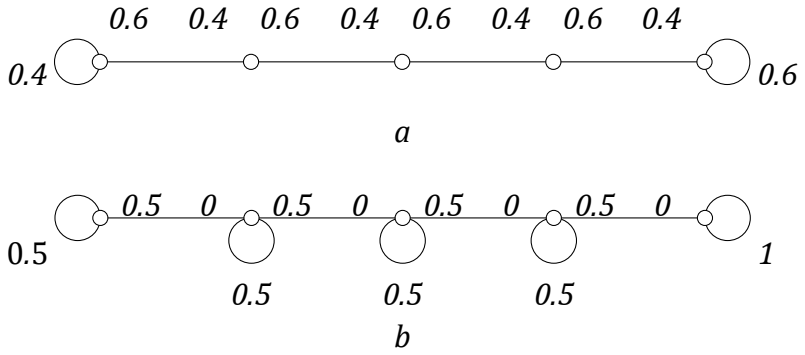
The use of normalized conductance to prove convergence of Markov Chains is by Sinclair and Jerrum, [SJ89] and Alon [Alo86]. A polynomial time bounded Markov chain based method for estimating the volume of convex sets was developed by Dyer, Frieze and Kannan [DFK91].

4.10 Exercises

Exercise 4.1 *The Fundamental Theorem of Markov chains says that for a connected Markov chain, the long-term average distribution* $\mathbf{a}(\mathbf{t})$ *converges to a stationary distribution. Does the t step distribution* $\mathbf{p}(\mathbf{t})$ *also converge for every connected Markov Chain? Consider the following examples: (i) A two-state chain with* $p_{12} = p_{21} = 1$*. (ii) A three state chain with* $p_{12} = p_{23} = p_{31} = 1$ *and the other* $p_{ij} = 0$*. Generalize these examples to produce Markov Chains with many states.*

Exercise 4.2 *Does* $\lim_{t\to\infty} a(t) - a(t+1) = 0$ *imply that* $a(t)$ *converges to some value? Hint: consider the average cumulative sum of the digits in the sequence* $10^2 1^4 0^8 1^{16} \cdots$

Exercise 4.3 *What is the stationary probability for the following networks.*



Exercise 4.4 *A Markov chain is said to be symmetric if for all i and j,* $p_{ij} = p_{ji}$*. What is the stationary distribution of a connected symmetric chain? Prove your answer.*

Exercise 4.5 *Prove* $|\mathbf{p} - \mathbf{q}|_1 = 2\sum_i (p_i - q_i)^+$ *for probability distributions* $\mathbf{p}$ *and* $\mathbf{q}$*, (Proposition 4.4).*

Exercise 4.6 *Let* $p(\mathbf{x})$*, where* $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ $x_i \in \{0,1\}$*, be a multivariate probability distribution. For* $d = 100$*, how would you estimate the marginal distribution*

$$p(x_1) = \sum_{x_2,\ldots,x_d} p(x_1, x_2, \ldots, x_d$$

)?

Exercise 4.7 *Using the Metropolis-Hasting Algorithm create a Markov chain whose stationary probability is that given in the following table. Use the* 3×3 *lattice for the underlying graph.*

x_1x_2	00	01	02	10	11	12	20	21	22
Prob	1/16	1/8	1/16	1/8	1/4	1/8	1/16	1/8	1/16

Exercise 4.8 *Using Gibbs sampling create a* 4×4 *lattice where vertices in rows and columns are cliques whose stationary probability is that given in the following table.*

x/y	1	2	3	4
1	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{16}$
2	$\frac{1}{32}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{32}$
3	$\frac{1}{32}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{32}$
4	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{16}$

Note by symmetry there are only three types of vertices and only two types of rows or columns.

Exercise 4.9 *How would you integrate a high dimensional multivariate polynomial distribution over some convex region?*

Exercise 4.10 *Given a time-reversible Markov chain, modify the chain as follows. At the current state, stay put (no move) with probability 1/2. With the other probability 1/2, move as in the old chain. Show that the new chain has the same stationary distribution. What happens to the convergence time in this modification?*

Exercise 4.11 *Let* **p** *be a probability vector (nonnegative components adding up to 1) on the vertices of a connected graph which is sufficiently large that it cannot be stored in a computer. Set* p_{ij} *(the transition probability from i to j) to* p_j *for all i* 6= *j which are adjacent in the graph. Show that the stationary probability vector is* **p***. Is a random walk an efficient way to sample according to a probability distribution that is close to* **p***? Think, for example, of the graph G being the n-dimensional hypercube with* 2^n *vertices, and* **p** *as the uniform distribution over those vertices.*

Exercise 4.12 *Construct the edge probability for a three state Markov chain where each pair of states is connected by an undirected edge so that the stationary probability is* $\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\big)$ *. Repeat adding a self loop with probability* $\frac{1}{2}$ *to the vertex with probability* $\frac{1}{2}$*.*

Exercise 4.13 *Consider a three state Markov chain with stationary probability* $\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\big)$*. Consider the Metropolis-Hastings algorithm with G the complete graph on these three vertices. For each edge and each direction what is the expected probability that we would actually make a move along the edge?*

Exercise 4.14 *Consider a distribution* **p** *over* $\{0,1\}^2$ *with* $p(00) = p(11) = \frac{1}{2}$ *and* $p(01) = p(10) = 0$*. Give a connected graph on* $\{0,1\}^2$ *that would be bad for running MetropolisHastings and a graph that would be good for running Metropolis-Hastings. What would be the problem with Gibbs sampling?*

Exercise 4.15 *Consider p*(**x**) *where* **x** ∈ {0,1}^{100} *such that* $p(\mathbf{0}) = \frac{1}{2}$ *and* $p(\mathbf{x}) = \frac{1/2}{(2^{100}-1)}$ *for* **x** 6= **0**. *How does Gibbs sampling behave?*

Exercise 4.16 *Given a connected graph G and an integer k how would you generate connected subgraphs of G with k vertices with probability proportional to the number of edges in the subgraph? A subgraph of G does not need to have all edges of G that join vertices of the subgraph. The probabilities need not be exactly proportional to the number of edges and you are not expected to prove your algorithm for this problem.*

Exercise 4.17 *Suppose one wishes to generate uniformly at random a regular, degree three, undirected, not necessarily connected multi-graph with 1,000 vertices. A multigraph may have multiple edges between a pair of vertices and self loops. One decides to do this by a Markov Chain Monte Carlo technique. In particular, consider a (very large) network where each vertex corresponds to a regular degree three, 1,000 vertex multi-graph. For edges, say that the vertices corresponding to two graphs are connected by an edge if one graph can be obtained from the other by a flip of a pair of edges. In a flip, a pair of edges* (*a,b*) *and* (*c,d*) *are replaced by* (*a,c*) *and* (*b,d*).

1. *Prove that the network whose vertices correspond to the desired graphs is connected. That is, for any two 1000-vertex degree-3 multigraphs, it is possible to walk from one to the other in this network.*

2. *Prove that the stationary probability of the random walk is uniform over all vertices.*

3. *Give an upper bound on the diameter of the network.*

4. *How would you modify the process if you wanted to uniformly generate connected degree three multi-graphs?*

In order to use a random walk to generate the graphs in a reasonable amount of time, the random walk must rapidly converge to the stationary probability. Proving this is beyond the material in this book.

Exercise 4.18 *Construct, program, and execute an algorithm to estimate the volume of a unit radius sphere in 20 dimensions by carrying out a random walk on a* 20 *dimensional grid with 0.1 spacing.*

Exercise 4.19 *What is the mixing time for the undirected graphs*

1. *Two cliques connected by a single edge?*

2. *A graph consisting of an n vertex clique plus one additional vertex connected to one vertex in the clique.*

Exercise 4.20 *What is the mixing time for*

1. *G*(*n,p*) *with* $p = \frac{\log n}{n}$?

2. *A circle with n vertices where at each vertex an edge has been added to another vertex chosen at random. On average each vertex will have degree four, two circle edges, and an edge from that vertex to a vertex chosen at random, and possible some edges that are the ends of the random edges from other vertices.*

Exercise 4.21 *Find the -mixing time for a 2-dimensional lattice with n vertices in each coordinate direction with a uniform probability distribution. To do this solve the following problems.*

1. *The minimum number of edges leaving a set S of size greater than or equal to* $n^2/4$ *is n.*

2. *The minimum number of edges leaving a set*√ *S of size less than or equal to* $n^2/4$ *is*

   b $\overline{Sc}$.

3. *Compute* $\Phi(S)$

4. *Compute* Φ

5. *Computer the -mixing time*

Exercise 4.22 *Find the -mixing time for a d-dimensional lattice with n vertices in each coordinate direction with a uniform probability distribution. To do this, solve the following problems.*

1. *Select a direction say* x_1 *and push all elements of S in each column perpendicular to* $x_1 = 0$ *as close to* $x_1 = 0$ *as possible. Prove that the number of edges leaving S is at least as large as the number leaving the modified version of S.*

2. *Repeat step one for each direction. Argue that for a direction say* x_1*, as* x_1 *gets larger a set in the perpendicular plane is contained in the previous set.*

3. *Optimize the arrangements of elements in the plane* $x_1 = 0$ *and move elements from farthest out plane in to make all planes the same shape as* $x_1 = 0$ *except for some leftover elements of S in the last plane. Argue that this does not increase the number of edges out.*

4. *What configurations might we end up with?*

5. *Argue that for a given size, S has at least as many edges as the modified version of S.*

6. *What is* $\Phi(S)$ *for a modified form S?*

7. *What is* Φ *for a d-dimensional lattice?*

8. *What is the -mixing time?*

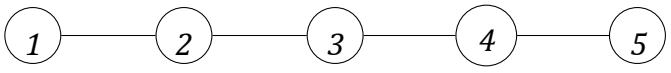
Exercise 4.23

1. *What is the set of possible harmonic functions on a connected graph if there are only interior vertices and no boundary vertices that supply the boundary condition?*
2. *Let q_x be the stationary probability of vertex x in a random walk on an undirected graph where all edges at a vertex are equally likely and let d_x be the degree of vertex x. Show that $\frac{q_x}{d_x}$ is a harmonic function.*
3. *If there are multiple harmonic functions when there are no boundary conditions, why is the stationary probability of a random walk on an undirected graph unique?*
4. *What is the stationary probability of a random walk on an undirected graph?*

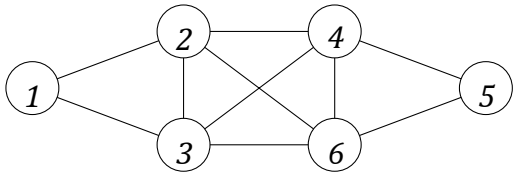
Exercise 4.24 *In Section 4.5, given an electrical network, we define an associated Markov chain such that voltages and currents in the electrical network corresponded to properties of the Markov chain. Can we go in the reverse order and for any Markov chain construct the equivalent electrical network?*

Exercise 4.25 *What is the probability of reaching vertex 1 before vertex 5 when starting a random walk at vertex 4 in each of the following graphs.*

1.



2.



Exercise 4.26 *Consider the electrical resistive network in Figure 4.14 consisting of vertices connected by resistors. Kirchoff's law states that the currents at each vertex sum to zero. Ohm's law states that the voltage across a resistor equals the product of the resistance times the current through it. Using these laws calculate the effective resistance of the network.*

Exercise 4.27 *Consider the electrical network of Figure 4.15.*

1. *Set the voltage at a to one and at b to zero. What are the voltages at c and d?*
2. *What is the current in the edges a to c, a to d, c to d. c to b and d to b?*

3. *What is the effective resistance between a and b?*

4. *Convert the electrical network to a graph. What are the edge probabilities at each vertex so that the probability of a walk starting at c (d) reaches a before b equals the voltage at c (the voltage at d).?*

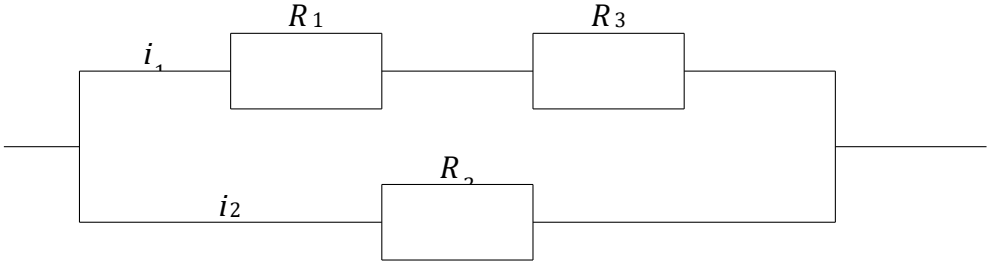


Figure 4.14: An electrical network of resistors.

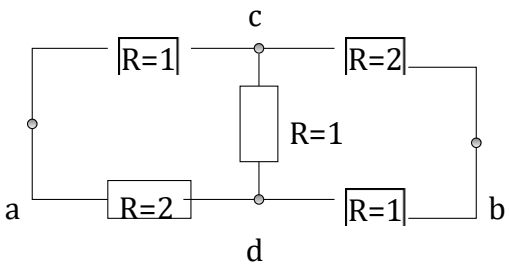


Figure 4.15: An electrical network of resistors.

5. *What is the probability of a walk starting at c reaching a before b? a walk starting at d reaching a before b?*

6. *What is the net frequency that a walk from a to b goes through the edge from c to d?*

7. *What is the probability that a random walk starting at a will return to a before reaching b?*

Exercise 4.28 *Consider a graph corresponding to an electrical network with vertices a and b. Prove directly that* $c_{c^{eff}_a}$ *must be less than or equal to one. We know that this is the escape probability and must be at most 1. But, for this exercise, do not use that fact.*

Exercise 4.29 *(Thomson's Principle) The energy dissipated by the resistance of edge xy in an electrical network is given by* $i^2_{xy}r_{xy}$*. The total energy dissipation in the network is* $E = \frac{1}{2}\sum_{x,y} i^2_{xy} r_{xy}$ *where the* $\frac{1}{2}$ *accounts for the fact that the dissipation in each edge is counted twice in the summation. Show that the actual current distribution is the distribution satisfying Ohm's law that minimizes energy dissipation.*

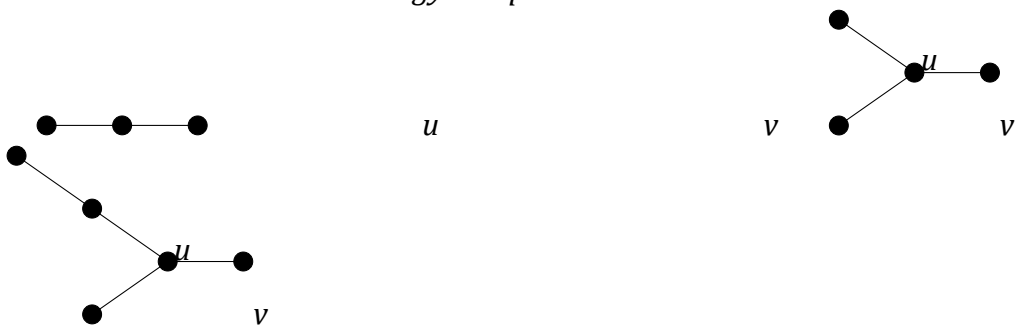


Figure 4.16: Three graphs

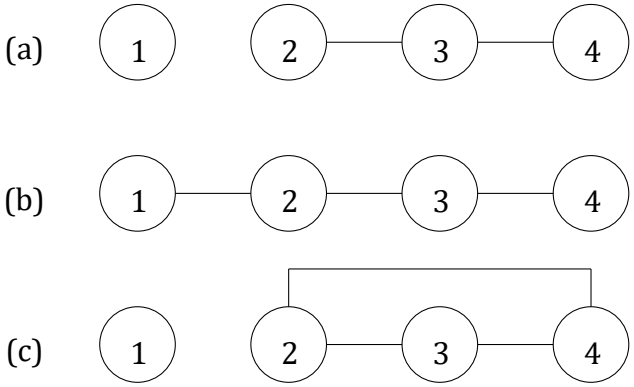


Figure 4.17: Three graph

Exercise 4.30 *(Rayleigh's law) Prove that reducing the value of a resistor in a network cannot increase the effective resistance. Prove that increasing the value of a resistor cannot decrease the effective resistance. You may use Thomson's principle Exercise 4.29.*

Exercise 4.31 *What is the hitting time* h_{uv} *for two adjacent vertices on a cycle of length n? What is the hitting time if the edge* (u,v) *is removed?*

Exercise 4.32 *What is the hitting time* h_{uv} *for the three graphs if Figure 4.16.*

Exercise 4.33 *Show that adding an edge can either increase or decrease hitting time by calculating h_{24} for the three graphs in Figure 4.17.*

Exercise 4.34 *Consider the n vertex connected graph shown in Figure 4.18 consisting of an edge* (u,v) *plus a connected graph on* $n-1$ *vertices and m edges. Prove that* $h_{uv} = 2m + 1$ *where m is the number of edges in the* $n-1$ *vertex subgraph.*

Exercise 4.35 *Consider a random walk on a clique of size n. What is the expected number of steps before a given vertex is reached?*

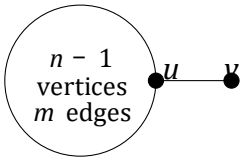


Figure 4.18: A connected graph consisting of $n-1$ vertices and m edges along with a single edge (u,v).

Exercise 4.36 *What is the most general solution to the difference equation* $t(i+2) - 5t(i+1) + 6t(i) = 0$*. How many boundary conditions do you need to make the solution unique?*

Exercise 4.37 *Given the difference equation* $a_k t(i+k) + a_{k-1}t(i+k-1) + \cdots + a_1 t(i+1) + a_0 t(i) = 0$ *the polynomial* $a_k t^k + a_{k-i}t^{k-1} + \cdots + a_1 t + a_0 = 0$ *is called the characteristic polynomial.*

1. *If the equation has a set of r distinct roots, what is the most general form of the solution?*
2. *If the roots of the characteristic polynomial are not distinct what is the most general form of the solution?*
3. *What is the dimension of the solution space?*
4. *If the difference equation is not homogeneous (i.e., the right hand side is not 0) and f(i) is a specific solution to the nonhomogeneous difference equation, what is the full set of solutions to the nonhomogeneous difference equation?*

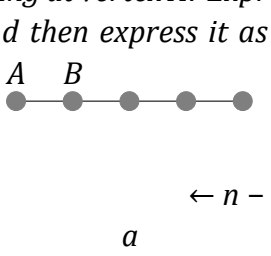
Exercise 4.38 *Show that adding an edge to a graph can either increase or decrease commute time.*

Exercise 4.39 *Consider the set of integers* $\{1,2,\ldots,n\}$.

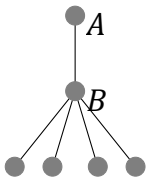
1. *What is the expected number of draws with replacement until the integer 1 is drawn.*

2. *What is the expected number of draws with replacement so that every integer is drawn?*

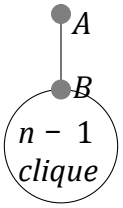
Exercise 4.40 *For each of the three graphs below what is the return time starting at vertex A? Express your answer as a function of the number of vertices, n, and then express it as a function of the number of edges m.*



a



b



c

Exercise 4.41 *Suppose that the clique in Exercise 4.40 was replaced by an arbitrary graph with m−1 edges. What would be the return time to A in terms of m, the total number of edges.*

Exercise 4.42 *Suppose that the clique in Exercise 4.40 was replaed by an arbitrary graph with m−d edges and there were d edges from A to the graph. What would be the expected length of a random path starting at A and ending at A after returning to A exactly d times.*

Exercise 4.43 *Given an undirected graph with a component consisting of a single edge find two eigenvalues of the Laplacian L = D−A where D is a diagonal matrix with vertex degrees on the diagonal and A is the adjacency matrix of the graph.*

Exercise 4.44 *A researcher was interested in determining the importance of various edges in an undirected graph. He computed the stationary probability for a random walk on the graph and let p_i be the probability of being at vertex i. If vertex i was of degree d_i, the frequency that edge (i,j) was traversed from i to j would be $\frac{1}{d_i}p_i$ and the frequency that the edge was traversed in the opposite direction would be $\frac{1}{d_j}p_j$. Thus, he assigned an importance of $\left|\frac{1}{d_i}p_i - \frac{1}{d_j}p_j\right|$ to the edge. What is wrong with his idea?*

Exercise 4.45 *Prove that two independent random walks starting at the origin on a two dimensional lattice will eventually meet with probability one.*

Exercise 4.46 *Suppose two individuals are flipping balanced coins and each is keeping tract of the number of heads minus the number of tails. At some time will both individual's counts be the same?*

Exercise 4.47 *Consider the lattice in 2-dimensions. In each square add the two diagonal edges. What is the escape probability for the resulting graph?*

Exercise 4.48 *Determine by simulation the escape probability for the 3-dimensional lattice.*

Exercise 4.49 *What is the escape probability for a random walk starting at the root of an infinite binary tree?*

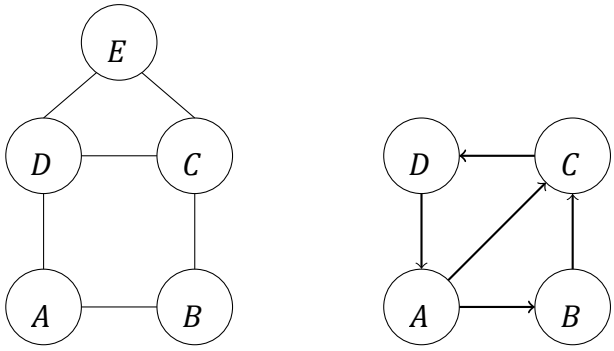


Figure 4.19: An undirected and a directed graph.

Exercise 4.50 *Consider a random walk on the positive half line, that is the integers* 0,1,2,.... *At the origin, always move right one step. At all other integers move right with probability 2/3 and left with probability 1/3. What is the escape probability?*

Exercise 4.51 *Consider the graphs in Figure 4.19. Calculate the stationary distribution for a random walk on each graph and the flow through each edge. What condition holds on the flow through edges in the undirected graph? In the directed graph?*

Exercise 4.52 *Create a random directed graph with 200 vertices and roughly eight edges per vertex. Add k new vertices and calculate the pagerank with and without directed edges from the k added vertices to vertex 1. How much does adding the k edges change the pagerank of vertices for various values of k and restart frequency? How much does adding a loop at vertex 1 change the pagerank? To do the experiment carefully one needs to consider the pagerank of a vertex to which the star is attached. If it has low pagerank its page rank is likely to increase a lot.*

Exercise 4.53 *Repeat the experiment in Exercise 4.52 for hitting time.*

Exercise 4.54 *Search engines ignore self loops in calculating pagerank. Thus, to increase pagerank one needs to resort to loops of length two. By how much can you increase the page rank of a page by adding a number of loops of length two?*

Exercise 4.55 *Number the vertices of a graph* {1,2,...,*n*}*. Define hitting time to be the expected time from vertex 1. In (2) assume that the vertices in the cycle are sequentially numbered.*

1. *What is the hitting time for a vertex in a complete directed graph with self loops?*

2. *What is the hitting time for a vertex in a directed cycle with n vertices?*
   Create exercise relating strongly connected and full rank Full
   rank implies strongly connected.
   Strongly connected does not necessarily imply full rank

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

Is graph aperiodic iff $\lambda_1 > \lambda_2$?

Exercise 4.56 *Using a web browser bring up a web page and look at the source html. How would you extract the url's of all hyperlinks on the page if you were doing a crawl of the web? With Internet Explorer click on "source" under "view" to access the html representation of the web page. With Firefox click on "page source" under "view".*

Exercise 4.57 *Sketch an algorithm to crawl the World Wide Web. There is a time delay between the time you seek a page and the time you get it. Thus, you cannot wait until the page arrives before starting another fetch. There are conventions that must be obeyed if one were to actually do a search. Sites specify information as to how long or which files can be searched. Do not attempt an actual search without guidance from a knowledgeable person.*

5 Machine Learning

5.1 Introduction

Machine learning algorithms are general purpose tools for generalizing from data. They have proven to be able to solve problems from many disciplines without detailed domain-specific knowledge. To date they have been highly successful for a wide range of tasks including computer vision, speech recognition, document classification, automated driving, computational science, and decision support.

The core problem. A core problem underlying many machine learning applications is learning a good classification rule from labeled data. This problem consists of a domain of interest X, called the *instance space*, such as the set of email messages or patient records, and a classification task, such as classifying email messages into spam versus non-spam or determining which patients will respond well to a given medical treatment. We will typically assume our instance space $X = \{0,1\}^d$ or $X = \mathbb{R}^d$, corresponding to data that is described by d Boolean or real-valued features. Features for email messages could be the presence or absence of various types of words, and features for patient records could be the results of various medical tests. To perform the learning task, our learning algorithm is given a set S of labeled *training examples*, which are points in X along with their correct classification. This training data could be a collection of email messages, each labeled as spam or not spam, or a collection of patients, each labeled by whether or not they responded well to the given medical treatment. Our algorithm then aims to use the training examples to produce a classification rule that will perform well over new data, i.e., new points in X. A key feature of machine learning, which distinguishes it from other algorithmic tasks, is that our goal is *generalization*: to use one set of data in order to perform well on new data we have not seen yet. We focus on *binary classification* where items in the domain of interest are classified into two categories (called the positive class and the negative class), as in the medical and spam-detection examples above, but nearly all the techniques described here will also apply to multi-way classification.

How to learn. A high-level approach that many algorithms we discuss will follow is to try to find a "simple" rule with good performance on the training data. For instance, in the case of classifying email messages, we might find a set of highly indicative words such that every spam email in the training data has at least one of these words and none of the non-spam emails has any of them; in this case, the rule "if the message has any of these words then it is spam, else it is not" would be a simple rule that performs well on the training data. Or, we might find a way of weighting words with positive and negative weights such that the total weighted sum of words in the email message is positive on the spam emails in the training data, and negative on the non-spam emails. We will then argue that so long as the training data is representative of what future data will look like, we can be confident

that any sufficiently "simple" rule that performs well on the training data will also perform well on future data. To make this into a formal math-

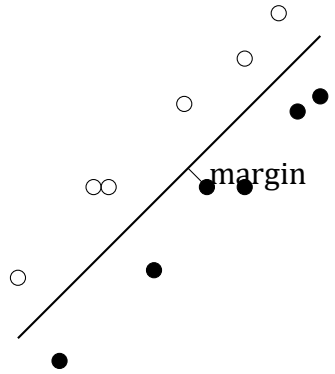


Figure 5.1: Margin of a linear separator.

ematical statement, we need to be precise about what we mean by "simple" as well as what it means for training data to be "representative" of future data. In fact, we will see several notions of complexity, including bit-counting and VC-dimension, that will allow us to make mathematical statements of this form. These statements can be viewed as formalizing the intuitive philosophical notion of Occam's razor.

5.2 The Perceptron algorithm

To help ground our discussion, we begin by describing a specific interesting learning algorithm, the *Perceptron algorithm*, for the problem of assigning positive and negative weights to features (such as words) so that each positive example has a positive sum of feature weights and each negative example has a negative sum of feature weights.

More specifically, the Perceptron algorithm is an efficient algorithm for finding a linear separator in d-dimensional space, with a running time that depends on the *margin of separation* of the data. We are given as input a set S of training examples (points in d-dimensional space), each labeled as positive or negative, and our assumption is that there exists a vector $\mathbf{w}^*$ such that for each positive example $\mathbf{x} \in S$ we have $\mathbf{x}^T\mathbf{w}^* \geq 1$ and for each negative example $\mathbf{x} \in S$ we have $\mathbf{x}^T\mathbf{w}^* \leq -1$. Note that the quantity $\mathbf{x}^T\mathbf{w}^*/|\mathbf{w}^*|$ is the distance of the point $\mathbf{x}$ to the hyperplane $\mathbf{x}^T\mathbf{w}^* = 0$. Thus, we can view our assumption as stating that there exists a linear separator through the origin with all positive examples on one side, all negative examples on the other side, and all examples at distance at least $\gamma = 1/|\mathbf{w}^*|$ from the separator. This quantity γ is called the *margin* of separation (see Figure 5.1).

The goal of the Perceptron algorithm is to find a vector $\mathbf{w}$ such that $\mathbf{x}^T\mathbf{w} > 0$ for all positive examples $\mathbf{x} \in S$, and $\mathbf{x}^T\mathbf{w} < 0$ for all negative examples $\mathbf{x} \in S$. It does so via the following update rule:

The Perceptron Algorithm: Start with the all-zeroes weight vector $\mathbf{w} = \mathbf{0}$. Then repeat the following until $\mathbf{x}^T\mathbf{w}$ has the correct sign for all $\mathbf{x} \in S$ (positive for positive examples and negative for negative examples):

1. Let $\mathbf{x} \in S$ be an example for which $\mathbf{x}^T\mathbf{w}$ does not have the correct sign.
2. Update as follows:
   (a) If $\mathbf{x}$ is a positive example, let $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}$.
   (b) If $\mathbf{x}$ is a negative example, let $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}$.

While simple, the Perceptron algorithm indeed will find a linear separator whenever one exists, making at most $(R/\gamma)^2$ updates where $R = \max_{\mathbf{x}\in S} |\mathbf{x}|$. Thus, if there exists a hyperplane through the origin that correctly separates the positive examples from the negative examples by a large margin relative to the radius of the smallest ball enclosing the data, then the total number of updates will be small.

Theorem 5.1 *If there exists a vector* $\mathbf{w}^*$ *such that* $\mathbf{x}^T\mathbf{w}^* \geq 1$ *for all positive examples* $\mathbf{x} \in S$ *and* $\mathbf{x}^T\mathbf{w}^* \leq -1$ *for all negative examples* $\mathbf{x} \in S$ *(i.e., a linear separator of margin* $\gamma = 1/|\mathbf{w}^*|$*), then the number of updates made by the Perceptron algorithm is at most* $R^2|\mathbf{w}^*|^2$*, where* $R = \max_{\mathbf{x}\in S} |\mathbf{x}|$.

To get a feel for this bound, notice that if we multiply all entries in all the $\mathbf{x} \in S$ by 100, we can divide all entries in $\mathbf{w}^*$ by 100 and it will still satisfy the "if"condition. So the bound is invariant to this kind of scaling, i.e., to our "units of measurement".

Proof of Theorem 5.1: Fix some $\mathbf{w}^*$ satisfying the "if" condition of the theorem. We will keep track of two quantities, $\mathbf{w}^T\mathbf{w}^*$ and $|\mathbf{w}|^2$. First of all, each time we make an update, $\mathbf{w}^T\mathbf{w}^*$ increases by at least 1. That is because if $\mathbf{x}$ is a positive example, then

$$(\mathbf{w} + \mathbf{x})^T\mathbf{w}^* = \mathbf{w}^T\mathbf{w}^* + \mathbf{x}^T\mathbf{w}^* \geq \mathbf{w}^T\mathbf{w}^* + 1,$$

by definition of $\mathbf{w}^*$. Similarly, if $\mathbf{x}$ is a negative example, then

$$(\mathbf{w} - \mathbf{x})^T\mathbf{w}^* = \mathbf{w}^T\mathbf{w}^* - \mathbf{x}^T\mathbf{w}^* \geq \mathbf{w}^T\mathbf{w}^* + 1.$$

Next, on each update, we claim that $|\mathbf{w}|^2$ increases by at most R^2. Let us first consider updates on positive examples. If we update on a positive example $\mathbf{x}$ then we have

$$(\mathbf{w} + \mathbf{x})^T(\mathbf{w} + \mathbf{x}) = |\mathbf{w}|^2 + 2\mathbf{x}^T\mathbf{w} + |\mathbf{x}|^2 \leq |\mathbf{w}|^2 + |\mathbf{x}|^2 \leq |\mathbf{w}|^2 + R^2,$$

where the middle inequality comes from the fact that we only perform an update on a positive example when $\mathbf{x}^T\mathbf{w} \leq 0$. Similarly, if we update on a negative example $\mathbf{x}$ then we have

$$(\mathbf{w} - \mathbf{x})^T(\mathbf{w} - \mathbf{x}) = |\mathbf{w}|^2 - 2\mathbf{x}^T\mathbf{w} + |\mathbf{x}|^2 \leq |\mathbf{w}|^2 + |\mathbf{x}|^2 \leq |\mathbf{w}|^2 + R^2.$$

Note that it is important here that we only update on examples for which $\mathbf{x}^T\mathbf{w}$ has the incorrect sign.

So, if we make M updates, then $\mathbf{w}^T\mathbf{w}^* \geq M$, and $|\mathbf{w}|^2 \leq MR^2$, or equivalently, $|\mathbf{w}| \leq R\sqrt{M}$. Finally, we use the fact that $\mathbf{w}^T\mathbf{w}^*/|\mathbf{w}^*| \leq |\mathbf{w}|$ which is just saying that the projection of $\mathbf{w}$ in the direction of $\mathbf{w}^*$ cannot be larger than the length of $\mathbf{w}$. This gives us:

$$\begin{aligned} M/|\mathbf{w}^*| &\leq R\sqrt{M} \\ \sqrt{M} &\leq R|\mathbf{w}^*| \\ M &\leq R^2|\mathbf{w}^*|^2 \end{aligned}$$

as desired. ∎

What if there is no $\mathbf{w}^*$ that *perfectly* separates the positive and negative examples? In Section 5.8 we will address this in the context of an online learning model, and we will see that the Perceptron algorithm enjoys strong guarantees even if the best $\mathbf{w}^*$ is not quite perfect, as a function of a quantity called the "hinge loss" of $\mathbf{w}^*$.

In the next section, we consider a related issue. Suppose the positive and negative examples are not *linearly* separable (there is no hyperplane with the positives on one side and the negatives on the other) but they are separable by some other simple curve such as a circle. In that case, we can use a technique known as *kernel functions*.

5.3 Kernel Functions

Suppose that instead of a linear separator decision boundary, the boundary between important emails and unimportant emails looks more like a circle, for example as in Figure 5.2.

A powerful idea for addressing situations like this is to use what are called *kernel functions*, or sometimes the "kernel trick". Here is the idea. Suppose you have a function K, called a "kernel", over pairs of data points such that for some function $\varphi : \mathsf{R}^d \to \mathsf{R}^N$, where perhaps $N \gg d$, we have $K(\mathbf{x},\mathbf{x}^0) = \varphi(\mathbf{x})^T\varphi(\mathbf{x}^0)$. In that case, if we can write the Perceptron algorithm so that it only interacts with the data via dot-products, and then replace every dot-product with an invocation of K, then we can act as if we had performed the function φ explicitly without having to actually compute φ.

For example, consider $K(\mathbf{x},\mathbf{x}^0) = (1+\mathbf{x}^T\mathbf{x}^0)^k$ for some integer $k \geq 1$. It turns out this corresponds to a mapping φ into a space of dimension $N \approx d^k$. For example, in the case $d = 2, k = 2$ we have (using x_i to denote the ith coordinate of $\mathbf{x}$):

$$\begin{aligned} K(\mathbf{x},\mathbf{x}^0) &= (1 + x_1x^0{}_1 + x_2x^0{}_2)^2 \\ &= 1 + 2x_1x_{01} + 2x_2x_{02} + x_{21}x_{012} + 2x_1x_2x_{01}x_{02} + x_{22}x_{022} \\ &= \varphi(\mathbf{x})^T\varphi(\mathbf{x}^0) \end{aligned}$$

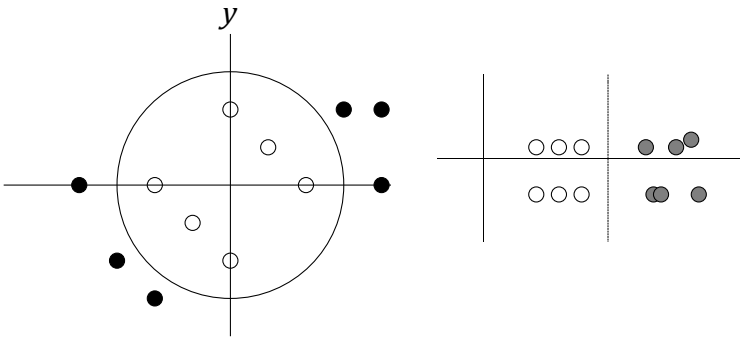


Figure 5.2: Data that is not linearly separable in the input space R^2 but that is linearly separable in the "φ-space," $\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)$, corresponding to the kernel function $K(\mathbf{x}, \mathbf{x}') = (1 + x_1x_1' + x_2x_2')^2$.

for $\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)$. Notice also that a linear separator in this space could correspond to a more complicated decision boundary such as an ellipse in the original space. For instance, the hyperplane $\varphi(\mathbf{x})^T\mathbf{w}^* = 0$ for $\mathbf{w}^* = (-4,0,0,1,0,1)$ corresponds to the circle $x_1^2 + x_2^2 = 4$ in the original space, such as in Figure 5.2.

The point of this is that if in the higher-dimensional "φ-space" there is a $\mathbf{w}^*$ such that the bound of Theorem 5.1 is small, then the algorithm will halt after not too many updates (and later we will see that under reasonable assumptions on the data, this implies we can be confident in its ability to perform well on new data as well). But the nice thing is we didn't have to computationally perform the mapping φ!

So, how can we view the Perceptron algorithm as only interacting with data via dotproducts? Notice that $\mathbf{w}$ is always a linear combination of data points. For example, if we made updates on examples $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_5$, and these examples were positive, positive, and negative respectively, we would have $\mathbf{w} = \mathbf{x}_1 + \mathbf{x}_2 - \mathbf{x}_5$. So, if we keep track of $\mathbf{w}$ this way, then to classify a new example $\mathbf{x}$, we can write $\mathbf{x}^T\mathbf{w} = \mathbf{x}^T\mathbf{x}_1 + \mathbf{x}^T\mathbf{x}_2 - \mathbf{x}^T\mathbf{x}_5$. So if we just replace each of these dot-products with "K", we are running the algorithm as if we had explicitly performed the φ mapping. This is called "kernelizing" the algorithm.

Many different pairwise functions on examples are legal kernel functions. One easy way to create a kernel function is by combining other kernel functions together, via the following theorem.

Theorem 5.2 *Suppose K_1 and K_2 are kernel functions. Then*

1. *For any constant $c \geq 0$, cK_1 is a legal kernel. In fact, for any scalar function f, the function $K_3(\mathbf{x},\mathbf{x}^0) = f(\mathbf{x})f(\mathbf{x}^0)K_1(\mathbf{x},\mathbf{x}^0)$ is a legal kernel.*

2. *The sum $K_1 + K_2$, is a legal kernel.*
3. *The product, K_1K_2, is a legal kernel.*

You will prove Theorem 5.2 in Exercise 5.9. Notice that this immediately implies that the function $K(\mathbf{x},\mathbf{x}^0) = (1 + \mathbf{x}^T\mathbf{x}^0)^k$ is a legal kernel by using the fact that $K_1(\mathbf{x},\mathbf{x}^0) = 1$ is a legal kernel, $K_2(\mathbf{x},\mathbf{x}^0) = \mathbf{x}^T\mathbf{x}^0$ is a legal kernel, then adding them, and then multiplying that by itself k times. Another popular kernel is the Gaussian kernel, defined as:

$$K(\mathbf{x},\mathbf{x}_0) = e^{-c|\mathbf{x}-\mathbf{x}_0|_2}.$$

If we think of a kernel as a measure of similarity, then this kernel defines the similarity between two data objects as a quantity that decreases exponentially with the squared distance between them. The Gaussian kernel can be shown to be a true kernel function by first writing it as $f(\mathbf{x})f(\mathbf{x}^0)e^{2c\mathbf{x}_T\mathbf{x}_0}$ for $f(\mathbf{x}) = e^{-c|\mathbf{x}|_2}$ and then taking the Taylor expansion of $e^{2c\mathbf{x}_T\mathbf{x}_0}$, applying the rules in Theorem 5.2. Technically, this last step requires considering countably infinitely many applications of the rules and allowing for infinite-dimensional vector spaces.

5.4 Generalizing to New Data

So far, we have focused on the problem of finding a classification rule that performs well on a given set S of training data. But what we really want our classification rule to do is to perform well on new data we have not seen yet. To make guarantees of this form, we need some assumption that our training data is somehow representative of what new data will look like; formally, we will assume they are drawn from the same probability distribution. Additionally, we will see that we will want our algorithm's classification rule to be "simple" in some way. Together, these two conditions will allow us to make *generalization guarantees*: guarantees on the ability of our learned classification rule to perform well on new unseen data.

Formalizing the problem. To formalize the learning problem, assume there is some probability distribution D over the instance space X, such that (a) our training set S consists of points drawn independently at random from D, and (b) our objective is to

predict well on new points that are also drawn from D. This is the sense in which we assume that our training data is representative of future data. Let c^*, called the *target concept*, denote the subset of X corresponding to the positive class for the binary classification we are aiming to make. For example, c^* would correspond to the set of all patients who respond well to a given treatment in a medical scenario, or it could correspond to the set of all spam emails in a spam-detection scenario. So, each point in our training set S is labeled according to whether or not it belongs to c^* and our goal is to produce a set $h \subseteq X$, called our *hypothesis*, which is close to c^* with respect to distribution D. The *true error* of h is $err_D(h) = \text{Prob}(h \triangle c^*)$ where "$\triangle$" denotes symmetric difference, and probability mass is according to D. In other words, the true error of h is the probability it incorrectly classifies a data point drawn at random from D. Our goal is to produce h of low true error. The *training error* of h, denoted $err_S(h)$, is the fraction of points in S on which h and c^* disagree. That is, $err_S(h) = |S \cap (h \triangle c^*)|/|S|$. Training error is also called *empirical error*. Note that even though S is assumed to consist of points randomly drawn from D, it is possible for a hypothesis h to have low training error or even to completely agree with c^* over the training sample, and yet have high true error. This is called *overfitting* the training data. For instance, a hypothesis h that simply consists of listing the positive examples in S, which is equivalent to a rule that memorizes the training sample and predicts positive on an example if and only if it already appeared positively in the training sample, would have zero training error. However, this hypothesis likely would have high true error and therefore would be highly overfitting the training data. More generally, overfitting is a concern because algorithms will typically be optimizing over the training sample. To design and analyze algorithms for learning, we will have to address the issue of overfitting.

To analyze overfitting, we introduce the notion of an hypothesis class, also called a concept class or set system. An hypothesis class H over X is a collection of subsets of X, called hypotheses. For instance, the class of *intervals* over $X = \mathbb{R}$ is the collection $\{[a,b] | a \leq b\}$. The class of *linear separators* over $X = \mathbb{R}^d$ is the collection

$$\{\{\mathbf{x} \in \mathbb{R}^d | \mathbf{w} \cdot \mathbf{x} \geq w_0\} | \mathbf{w} \in \mathbb{R}^d, w_0 \in \mathbb{R}\};$$

that is, it is the collection of all sets in $\mathbb{R}^d$ that are linearly separable from their complement. In the case that X is the set of 4 points in the plane $\{(-1,-1),(-1,1),(1,-1),(1,1)\}$, the class of linear separators contains 14 of the $2^4 = 16$ possible subsets of X.[17] Given an hypothesis class H and training set S, what we typically aim to do algorithmically is to find the hypothesis in H that most closely agrees with c^* over S. For example, we saw that the Perceptron algorithm will find a linear separator that agrees with the target function over S so long as S is linearly separable. To address overfitting, we argue that if S is large enough compared to some property of H, then with high probability all $h \in H$ have their training

[17] The only two subsets that are not in the class are the sets $\{(-1,-1),(1,1)\}$ and $\{(-1,1),(1,-1)\}$.

error close to their true error, so that if we find a hypothesis whose training error is low, we can be confident its true error will be low as well.

Before giving our first result of this form, we note that it will often be convenient to associate each hypotheses with its {−1,1}-valued indicator function

$$h(x) = \begin{cases} 1 & x \in h \\ -1 & x \notin h \end{cases}$$

In this notation the true error of h is $err_D(h) = \text{Prob}_{x\sim D}[h(x) \neq c^*(x)]$ and the training error is $err_S(h) = \text{Prob}_{x\sim S}[h(x) \neq c^*(x)]$.

5.5 Overfitting and Uniform Convergence

We now present two generalization guarantees that explain how one can guard against overfitting. To keep things simple, we assume our hypothesis class H is *finite*. Later, we will see how to extend these results to infinite classes as well. Given a class of hypotheses H, the first result states that for any given greater than zero, so long as the training data set is large compared to $\frac{1}{\epsilon}\ln(|\mathcal{H}|)$, it is unlikely any hypothesis $h \in$ H will have zero training error but have true error greater than . This means that with high probability, any hypothesis that our algorithms finds that agrees with the target hypothesis on the training data will have low true error. The second result states that if the training data set is large compared to $\frac{1}{\epsilon^2}\ln(|\mathcal{H}|)$, then it is unlikely that the training error and true error will differ by more than for any hypothesis in H. This means that if we find an hypothesis in H whose training error is low, we can be confident its true error will be low as well, even if its training error is not zero.

The basic idea is the following. If we consider some h with large true error, and we select an element $x \in$ X at random according to D, there is a reasonable chance that x will belong to the symmetric difference $h \Delta c^*$. If we select a large enough training sample S with each point drawn independently from X according to D, the chance that S is completely disjoint from $h \Delta c^*$ will be incredibly small. This is just for a single hypothesis h but we can now apply the union bound over all $h \in$ H of large true error, when H is finite. We formalize this below.

Theorem 5.3 *Let* H *be an hypothesis class and let and δ be greater than zero. If a training set S of size*

$$n \geq \frac{1}{\epsilon}\big(\ln|\mathcal{H}| + \ln(1/\delta)\big),$$

is drawn from distribution D, then with probability greater than or equal to $1 - \delta$ *every h in* H *with true error* $err_D(h) \geq \epsilon$ *has training error* $err_S(h) > 0$*. Equivalently, with probability greater than or equal to* $1 - \delta$*, every* $h \in$ H *with training error zero has true error less than .*

Proof: Let $h_1, h_2, \ldots$ be the hypotheses in H with true error greater than or equal to . These are the hypotheses that we don't want to output. Consider drawing the sample S of size n and let A_i be the event that h_i is consistent with S. Since every h_i has true error greater than or equal to

$$\text{Prob}(A_i) \leq (1-\epsilon)^n.$$

In other words, if we fix h_i and draw a sample S of size n, the chance that h_i makes no mistakes on S is at most the probability that a coin of bias comes up tails n times in a row, which is $(1-\epsilon)^n$. By the union bound over all i we have

$$\text{Prob}(\cup_i A_i) \leq |\mathcal{H}|(1-\epsilon)^n.$$

Using the fact that $(1-\epsilon) \leq e^{-\epsilon}$, the probability that any hypothesis in H with true error greater than or equal to has training error zero is at most $|\mathcal{H}|e^{-\epsilon n}$. Replacing n by the sample size bound from the theorem statement, this is at most $|\text{H}|e^{-\ln|\text{H}|-\ln(1/\delta)} = \delta$ as desired. ■

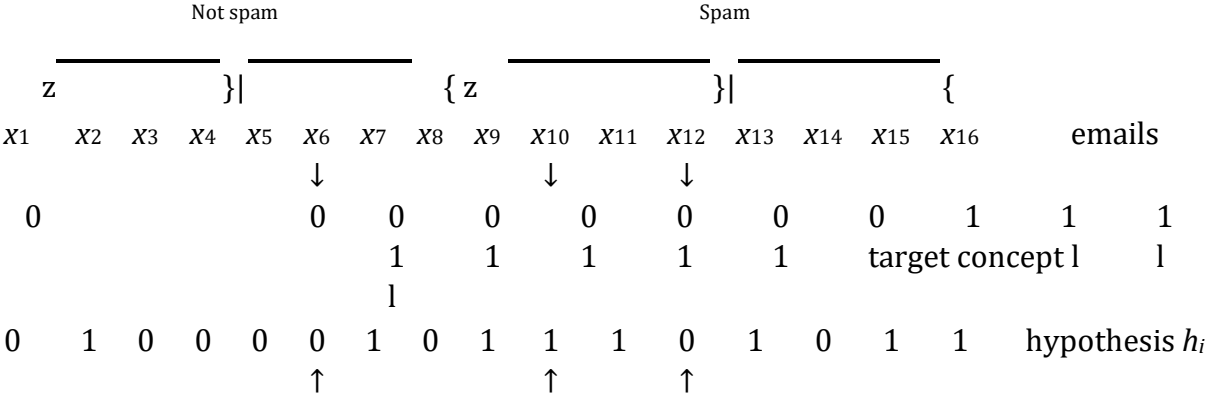


Figure 5.3: The hypothesis h_i disagrees with the truth in one quarter of the emails. Thus with a training set $|S|$, the probability that the hypothesis will survive is $(1 - 0.25)^{|S|}$

The conclusion of Theorem 5.3 is sometimes called a "PAC-learning guarantee" since it states that if we can find an $h \in$ H consistent with the sample, then this h is *P*robably *A*pproximately *C*orrect.

Theorem 5.3 addressed the case where there exists a hypothesis in H with zero training error. What if the best h_i in H has 5% error on S? Can we still be confident that its true error is low, say at most 10%? For this, we want an analog of Theorem 5.3 that says for a sufficiently large training set S, every $h_i \in$ H has training error within ± of the true error with high probability. Such a statement is called *uniform convergence* because we are asking that the training set errors converge to their true errors uniformly over all sets in H. To see intuitively why such a statement should be true for sufficiently large S and a single hypothesis h_i, consider two strings that differ in 10% of the positions and randomly

select a large sample of positions. The number of positions that differ in the sample will be close to 10%.

To prove uniform convergence bounds, we use a tail inequality for sums of independent Bernoulli random variables (i.e., coin tosses). The following is particularly convenient and is a variation on the Chernoff bounds in Section 12.6.1 of the appendix.

Theorem 5.4 (Hoeffding bounds) *Let $x_1,x_2,\ldots,x_n$ be independent* {0,1}*-valued random variables with Prob*(x_i = 1) = *p. Let s* = $\sum_i x_i$ *(equivalently, flip n coins of bias p and let s be the total number of heads). For any* $0 \le \alpha \le 1$,

$$Prob(s/n > p + \alpha) \le e^{-2n\alpha_2}$$

$$Prob(s/n < p - \alpha) \le e^{-2n\alpha_2}.$$

Theorem 5.4 implies the following uniform convergence analog of Theorem 5.3.

Theorem 5.5 (Uniform convergence) *Let* H *be a hypothesis class and let and δ be greater than zero. If a training set S of size*

$$n \ge \frac{1}{2\epsilon^2}\left(\ln|\mathcal{H}| + \ln(2/\delta)\right),$$

is drawn from distribution D, then with probability greater than or equal to 1−*δ, every h in* H *satisfies* $|err_S(h) - err_D(h)| \le \epsilon$.

Proof: First, fix some $h \in$ H and let x_j be the indicator random variable for the event that *h* makes a mistake on the j^{th} example in *S*. The x_j are independent {0,1} random variables and the probability that x_i equals 1 is the true error of *h*, and the fraction of the x_j's equal to 1 is exactly the training error of *h*. Therefore, Hoeffding bounds guarantee that the probability of the event A_h that $|err_D(h) - err_S(h)| > \epsilon$ is less than or equal to $2e^{-2n\epsilon^2}$. Applying the union bound to the events A_h over all $h \in$ H, the probability that there *exists* an $h \in$ H with the difference between true error and empirical error greater than is less than or equal to $2|\mathcal{H}|e^{-2n\epsilon^2}$. Using the value of *n* from the theorem statement, the right-hand-side of the above inequality is at most *δ* as desired. ■

Theorem 5.5 justifies the approach of optimizing over our training sample *S* even if we are not able to find a rule of zero training error. If our training set *S* is sufficiently large, with high probability, good performance on *S* will translate to good performance on *D*.

Note that Theorems 5.3 and 5.5 require |H| to be finite in order to be meaningful. The notion of growth functions and VC-dimension in Section 5.11 extend Theorem 5.5 to certain infinite hypothesis classes.

5.6 Illustrative Examples and Occam's Razor

We now present some examples to illustrate the use of Theorem 5.3 and 5.5 and also use these theorems to give a formal connection to the notion of Occam's razor.

5.6.1 Learning Disjunctions

Consider the instance space $X = \{0,1\}^d$ and suppose we believe that the target concept can be represented by a *disjunction* (an OR) over features, such as $c^* = \{x | x_1 = 1 \vee x_4 = 1 \vee x_8 = 1\}$, or more succinctly, $c^* = x_1 \vee x_4 \vee x_8$. For example, if we are trying to predict whether an email message is spam or not, and our features correspond to the presence or absence of different possible indicators of spam-ness, then this would correspond to the belief that there is some subset of these indicators such that every spam email has at least one of them and every non-spam email has none of them. Formally, let H denote the class of disjunctions, and notice that $|H| = 2^d$. So, by Theorem 5.3, it suffices to find a consistent disjunction over a sample S of size

$$|S| = \frac{1}{\epsilon}\left(d\ln(2) + \ln(1/\delta)\right).$$

How can we efficiently find a consistent disjunction when one exists? Here is a simple algorithm.

Simple Disjunction Learner: Given sample S, discard all features that are set to 1 in any negative example in S. Output the concept h that is the OR of all features that remain.

Lemma 5.6 *The Simple Disjunction Learner produces a disjunction h that is consistent with the sample S (i.e., with* $err_S(h) = 0$*) whenever the target concept is indeed a disjunction.*

Proof: Suppose target concept c^* is a disjunction. Then for any x_i that is listed in c^*, x_i will not be set to 1 in any negative example by definition of an OR. Therefore, h will include x_i as well. Since h contains all variables listed in c^*, this ensures that h will correctly predict positive on all positive examples in S. Furthermore, h will correctly predict negative on all negative examples in S since by design all features set to 1 in any negative example were discarded. Therefore, h is correct on all examples in S. ■

Thus, combining Lemma 5.6 with Theorem 5.3, we have an efficient algorithm for PAC-learning the class of disjunctions.

5.6.2 Occam's Razor

Occam's razor is the notion, stated by William of Occam around AD 1320, that in general one should prefer simpler explanations over more complicated ones.[18] Why should one

[18] The statement more explicitly was that "Entities should not be multiplied unnecessarily."

do this, and can we make a formal claim about why this is a good idea? What if each of us disagrees about precisely which explanations are simpler than others? It turns out we can use Theorem 5.3 to make a mathematical statement of Occam's razor that addresses these issues.

First, what do we mean by a rule being "simple"? Let's assume that each of us has some way of describing rules, using bits (since we are computer scientists). The methods, also called *description languages*, used by each of us may be different, but one fact we can say for certain is that in any given description language, there are at most 2^b rules that can be described using fewer than b bits (because $1 + 2 + 4 + \ldots + 2^{b-1} < 2^b$). Therefore, by setting H to be the set of all rules that can be described in fewer than b bits and plugging into Theorem 5.3, we have the following:

Theorem 5.7 (Occam's razor) *Fix any description language, and consider a training sample S drawn from distribution* D*. With probability at least* $1 - \delta$*, any rule h with* $err_S(h) = 0$ *that can be described using fewer than b bits will have* $err_D(h) \leq \epsilon$ *for* $|S| = \frac{1}{\epsilon}[b\ln(2) + \ln(1/\delta)]$*. Equivalently, with probability at least* $1 - \delta$*, all rules with* $err_S(h) = 0$ *that can be described in fewer than b bits will have* $err_D(h) \leq \frac{b\ln(2)+\ln(1/\delta)}{|S|}$.

For example, using the fact that $\ln(2) < 1$ and ignoring the low-order $\ln(1/\delta)$ term, this means that if the number of bits it takes to write down a rule consistent with the training data is at most 10% of the number of data points in our sample, then we can be confident

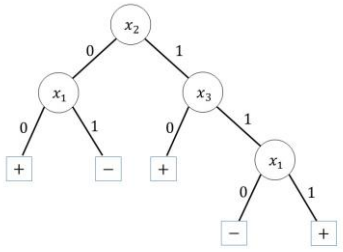


Figure 5.4: A decision tree with three internal nodes and four leaves. This tree corresponds to the Boolean function $\bar{x}_1\bar{x}_2 \vee x_1x_2x_3 \vee x_2\bar{x}_3$.

it will have error at most 10% with respect to D. What is perhaps surprising about this theorem is that it means that we can each have different ways of describing rules and yet all use Occam's razor. Note that the theorem does not say that complicated rules are necessarily bad, or even that given two rules consistent with the data that the complicated rule is necessarily worse. What it does say is that Occam's razor is a good policy in that simple rules are unlikely to fool us since there are just not that many simple rules.

5.6.3 Application: Learning Decision Trees

One popular practical method for machine learning is to learn a *decision tree*; see Figure 5.4. While finding the smallest decision tree that fits a given training sample S is NPhard, there are a number of heuristics that are used in practice.[19] Suppose we run such a heuristic on a training set S and it outputs a tree with k nodes. Such a tree can be described using $O(k \log d)$ bits: $\log_2(d)$ bits to give the index of the feature in the root, $O(1)$ bits to indicate for each child if it is a leaf and if so what label it should have, and then $O(k_L \log d)$ and $O(k_R \log d)$ bits respectively to describe the left and right subtrees, where k_L is the number of nodes in the left subtree and k_R is the number of nodes in the right subtree. So, by Theorem 5.7, we can be confident the true error is low if we can produce a consistent tree with fewer than $\epsilon|S|/\log(d)$ nodes.

5.7 Regularization: Penalizing Complexity

Theorems 5.5 and 5.7 suggest the following idea. Suppose that there is no simple rule that is perfectly consistent with the training data, but we notice there are very simple rules with training error 20%, say, and then some more complex rules with training error 10%, and so on. In this case, perhaps we should optimize some combination of training error and simplicity. This is the notion of *regularization*, also called *complexity penalization*.

Specifically, a *regularizer* is a penalty term that penalizes more complex hypotheses. Given our theorems so far, a natural measure of complexity of a hypothesis is the number of bits we need to write it down.[20] Consider now fixing some description language, and let H_i denote those hypotheses that can be described in i bits in this language, so $|H_i| \leq 2^i$. Let $\delta_i = \delta/2^i$. Rearranging the bound of Theorem 5.5, we know that with probability at least $1 - \delta_i$, all $h \in H_i$ satisfy $err_D(h) \leq err_S(h) + \sqrt{\frac{\ln(|\mathcal{H}_i|)+\ln(2/\delta_i)}{2|S|}}$. Now, applying the union bound over all i, using the fact that $\delta_1 + \delta_2 + \delta_3 + \ldots = \delta$, and also the fact that $\ln(|H_i|) + \ln(2/\delta_i) \leq i\ln(4) + \ln(2/\delta)$, gives the following corollary.

Corollary 5.8 *Fix any description language, and consider a training sample S drawn from distribution* D*. With probability greater than or equal to* $1 - \delta$*, all hypotheses h satisfy*

[19] For instance, one popular heuristic, called ID3, selects the feature to put inside any given node v by choosing the feature of largest *information gain*, a measure of how much it is directly improving prediction. Formally, using S_v to denote the set of examples in S that reach node v, and supposing that feature x_i partitions S_v into S_v^0 and S_v^1 (the examples in S_v with $x_i = 0$ and $x_i = 1$, respectively), the information gain of x_i is defined as: $Ent(S_v) - [\frac{|S_v^0|}{|S_v|}Ent(S_v^0) + \frac{|S_v^1|}{|S_v|}Ent(S_v^1)]$. Here, $Ent(S^0)$ is the binary entropy of the label proportions in set S^0; that is, if a p fraction of the examples in S^0 are positive, then $Ent(S^0) = p\log_2(1/p)+(1-p)\log_2(1/(1-p))$, defining $0\log_2(0) = 0$. This then continues until all leaves are pure—they have only positive or only negative examples.

[20] Later we will see support vector machines that use a regularizer for linear separators based on the margin of separation of data.

$$err_D(h) \leq err_S(h) + \sqrt{\frac{\text{size}(h)\ln(4) + \ln(2/\delta)}{2|S|}}$$

where size(h) *denotes the number of bits needed to describe h in the given language.*

Corollary 5.8 gives us the tradeoff we were looking for. It tells us that rather than searching for a rule of low training error, we instead may want to search for a rule with a low right-hand-side in the displayed formula. If we can find one for which this quantity is small, we can be confident true error will be low as well.

5.8 Online Learning

So far we have been considering what is often called the *batch learning* scenario. You are given a "batch" of data—the training sample S—and your goal is to use it to produce a hypothesis h that will have low error on new data, under the assumption that both S and the new data are sampled from some fixed distribution D. We now switch to the more challenging *online learning* scenario where we remove the assumption that data is sampled from a fixed probability distribution, or from any probabilistic process at all.

Specifically, the online learning scenario proceeds as follows. At each time t = 1,2,..., two events occur:

1. The algorithm is presented with an arbitrary example $x_t \in X$ and is asked to make a prediction $\grave{}_t$ of its label.

2. The algorithm is told the true label of the example $c^*(x_t)$ and is charged for a mistake if $c^*(x_t) 6= \grave{}_t$.

The goal of the learning algorithm is to make as few mistakes as possible in total. For example, consider an email classifier that when a new email message arrives must classify it as "important" or "it can wait". The user then looks at the email and informs the algorithm if it was incorrect. We might not want to model email messages as independent random objects from a fixed probability distribution, because they often are replies to previous emails and build on each other. Thus, the online learning model would be more appropriate than the batch model for this setting.

Intuitively, the online learning model is harder than the batch model because we have removed the requirement that our data consists of independent draws from a fixed probability distribution. Indeed, we will see shortly that any algorithm with good performance in the online model can be converted to an algorithm with good performance in the batch model. Nonetheless, the online model can sometimes be a cleaner model for design and analysis of algorithms.

5.8.1 An Example: Learning Disjunctions

As a simple example, let's revisit the problem of learning disjunctions in the online model. We can solve this problem by starting with a hypothesis $h = x_1 \vee x_2 \vee \ldots \vee x_d$ and using it for prediction. We will maintain the invariant that every variable in the target disjunction is also in our hypothesis, which is clearly true at the start. This ensures that the only mistakes possible are on examples x for which $h(x)$ is positive but $c^*(x)$ is negative. When such a mistake occurs, we simply remove from h any variable set to 1 in x. Since such variables cannot be in the target function (since x was negative), we maintain our invariant *and* remove at least one variable from h. This implies that the algorithm makes at most d mistakes total on any series of examples consistent with a disjunction.

In fact, we can show this bound is tight by showing that no deterministic algorithm can guarantee to make fewer than d mistakes.

Theorem 5.9 *For any deterministic algorithm A there exists a sequence of examples σ and disjunction c^* such that A makes at least d mistakes on sequence σ labeled by c^*.*

Proof: Let σ be the sequence $e_1, e_2, \ldots, e_d$ where e_j is the example that is zero everywhere except for a 1 in the jth position. Imagine running A on sequence σ and telling A it made a mistake on every example; that is, if A predicts positive on e_j we set $c^*(e_j) = -1$ and if A predicts negative on e_j we set $c^*(e_j) = +1$. This target corresponds to the disjunction of all x_j such that A predicted negative on e_j, so it is a legal disjunction. Since A is deterministic, the fact that we constructed c^* by running A is not a problem: it would make the same mistakes if re-run from scratch on the same sequence and same target. Therefore, A makes d mistakes on this σ and c^*. ■

5.8.2 The Halving Algorithm

If we are not concerned with running time, a simple algorithm that guarantees to make at most $\log_2(|H|)$ mistakes for a target belonging to any given class H is called the *halving algorithm*. This algorithm simply maintains the *version space* $V \subseteq H$ consisting of all $h \in H$ consistent with the labels on every example seen so far, and predicts based on majority vote over these functions. Each mistake is guaranteed to reduce the size of the version space V by at least half (hence the name), thus the total number of mistakes is at most $\log_2(|H|)$. Note that this can be viewed as the number of bits needed to write a function in H down.

5.8.3 The Perceptron Algorithm

Earlier we described the Perceptron algorithm as a method for finding a linear separator consistent with a given training set S. However, the Perceptron algorithm also operates naturally in the online setting as well.

Recall that the basic assumption of the Perceptron algorithm is that the target function can be described by a vector $\mathbf{w}^*$ such that for each positive example $\mathbf{x}$ we have $\mathbf{x}^T\mathbf{w}^* \geq 1$ and for each negative example $\mathbf{x}$ we have $\mathbf{x}^T\mathbf{w}^* \leq -1$. Recall also that we can interpret $\mathbf{x}^T\mathbf{w}^*/|\mathbf{w}^*|$ as the distance of $\mathbf{x}$ to the hyperplane $\mathbf{x}^T\mathbf{w}^* = 0$. Thus, we can view our assumption as stating that there exists a linear separator through the origin with all positive examples on one side, all negative examples on the other side, and all examples at distance at least $\gamma = 1/|\mathbf{w}^*|$ from the separator, where γ is called the margin of separation.

The guarantee of the Perceptron algorithm will be that the total number of mistakes is at most $(R/\gamma)^2$ where $R = \max_t |\mathbf{x}_t|$ over all examples $\mathbf{x}_t$ seen so far. Thus, if there exists a hyperplane through the origin that correctly separates the positive examples from the negative examples by a large margin relative to the radius of the smallest ball enclosing the data, then the total number of mistakes will be small. The algorithm, restated in the online setting, is as follows.

The Perceptron Algorithm: Start with the all-zeroes weight vector $\mathbf{w} = \mathbf{0}$. Then, for $t = 1,2,\ldots$ do:

1. Given example $\mathbf{x}_t$, predict $\text{sgn}(\mathbf{x}^T_t\mathbf{w})$.

2. If the prediction was a mistake, then update:

   (a) If $\mathbf{x}_t$ was a positive example, let $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_t$.

   (b) If $\mathbf{x}_t$ was a negative example, let $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}_t$.

The Perceptron algorithm enjoys the following guarantee on its total number of mistakes.

Theorem 5.10 *On any sequence of examples $\mathbf{x}_1,\mathbf{x}_2,\ldots$, if there exists a vector $\mathbf{w}^*$ such that $\mathbf{x}^T_t\mathbf{w}^* \geq 1$ for the positive examples and $\mathbf{x}^T_t\mathbf{w}^* \leq -1$ for the negative examples (i.e., a linear separator of margin $\gamma = 1/|\mathbf{w}^*|$), then the Perceptron algorithm makes at most $R^2|\mathbf{w}^*|^2$ mistakes, where $R = \max_t |\mathbf{x}_t|$.*

Proof: Fix some consistent $\mathbf{w}^*$. We will keep track of two quantities, $\mathbf{w}^T\mathbf{w}^*$ and $|\mathbf{w}|^2$. First of all, each time we make a mistake, $\mathbf{w}^T\mathbf{w}^*$ increases by at least 1. That is because if $\mathbf{x}_t$ is a positive example, then

$$(\mathbf{w} + \mathbf{x}_t)^T\mathbf{w}^* = \mathbf{w}^T\mathbf{w}^* + \mathbf{x}_t^T\mathbf{w}^* \geq \mathbf{w}^T\mathbf{w}^* + 1,$$

by definition of $\mathbf{w}^*$. Similarly, if $\mathbf{x}_t$ is a negative example, then

$$(\mathbf{w} - \mathbf{x}_t)^T\mathbf{w}^* = \mathbf{w}^T\mathbf{w}^* - \mathbf{x}_t^T\mathbf{w}^* \geq \mathbf{w}^T\mathbf{w}^* + 1.$$

Next, on each mistake, we claim that $|\mathbf{w}|^2$ increases by at most R^2. Let us first consider mistakes on positive examples. If we make a mistake on a positive example $\mathbf{x}_t$ then we have

$$(\mathbf{w}+\mathbf{x}_t)^T(\mathbf{w}+\mathbf{x}_t) = |\mathbf{w}|^2 + 2\mathbf{x}_t^T\mathbf{w} + |\mathbf{x}_t|^2 \leq |\mathbf{w}|^2 + |\mathbf{x}_t|^2 \leq |\mathbf{w}|^2 + R^2,$$

where the middle inequality comes from the fact that we made a mistake, which means that $\mathbf{x}_t^T\mathbf{w} \leq 0$. Similarly, if we make a mistake on a negative example $\mathbf{x}_t$ then we have

$$(\mathbf{w}-\mathbf{x}_t)^T(\mathbf{w}-\mathbf{x}_t) = |\mathbf{w}|^2 - 2\mathbf{x}_t^T\mathbf{w} + |\mathbf{x}_t|^2 \leq |\mathbf{w}|^2 + |\mathbf{x}_t|^2 \leq |\mathbf{w}|^2 + R^2.$$

Note that it is important here that we only update on a mistake.

So, if we make M mistakes, then $\mathbf{w}^T\mathbf{w}^* \geq M$, and $|\mathbf{w}|^2 \leq MR^2$, or equivalently, $|\mathbf{w}| \leq R\sqrt{M}$. Finally, we use the fact that $\mathbf{w}^T\mathbf{w}^*/|\mathbf{w}^*| \leq |\mathbf{w}|$ which is just saying that the projection of $\mathbf{w}$ in the direction of $\mathbf{w}^*$ cannot be larger than the length of $\mathbf{w}$. This gives us:

$$\begin{aligned} M/|\mathbf{w}^*| &\leq R\sqrt{M} \\ \sqrt{M} &\leq R|\mathbf{w}^*| \\ M &\leq R^2|\mathbf{w}^*|^2 \end{aligned}$$

as desired. ∎

5.8.4 Extensions: Inseparable Data and Hinge Loss

We assumed above that there exists a perfect $\mathbf{w}^*$ that correctly classifies all the examples, e.g., correctly classifies all the emails into important versus non-important. This is rarely the case in real-life data. What if even the best $\mathbf{w}^*$ isn't quite perfect? We can see what this does to the above proof: if there is an example that $\mathbf{w}^*$ doesn't correctly classify, then while the second part of the proof still holds, the first part (the dot product of $\mathbf{w}$ with $\mathbf{w}^*$ increasing) breaks down. However, if this doesn't happen too often, and also $\mathbf{x}^T_t\mathbf{w}^*$ is just a "little bit wrong" then we will only make a few more mistakes.

To make this formal, define the *hinge-loss* of $\mathbf{w}^*$ on a positive example $\mathbf{x}_t$ as $\max(0, 1-\mathbf{x}_t^T\mathbf{w}^*)$. In other words, if $\mathbf{x}_t^T\mathbf{w}^* \geq 1$ as desired then the hinge-loss is zero; else, the hingeloss is the amount the LHS is less than the RHS.[21] Similarly, the hinge-loss of $\mathbf{w}^*$ on a negative example $\mathbf{x}_t$ is $\max(0, 1+\mathbf{x}^T_t\mathbf{w}^*)$. Given a sequence of labeled examples S, define the total hinge-loss $L_{hinge}(\mathbf{w}^*, S)$ as the sum of hinge-losses of $\mathbf{w}^*$ on all examples in S. We now get the following extended theorem.

[21] This is called "hinge-loss" because as a function of $\mathbf{x}_t^T\mathbf{w}^*$ it looks like a hinge.

Theorem 5.11 *On any sequence of examples* $S = \mathbf{x}_1, \mathbf{x}_2, \ldots$*, the Perceptron algorithm makes at most*

$$\min_{\mathbf{w}^*} \left(R^2 |\mathbf{w}^*|^2 + 2L_{hinge}(\mathbf{w}^*, S) \right)$$

mistakes, where $R = \max_t |\mathbf{x}_t|$.

Proof: As before, each update of the Perceptron algorithm increases $|\mathbf{w}|^2$ by at most R^2, so if the algorithm makes M mistakes, we have $|\mathbf{w}|^2 \leq MR^2$.

What we can no longer say is that each update of the algorithm increases $\mathbf{w}^T \mathbf{w}^*$ by at least 1. Instead, on a positive example we are "increasing" $\mathbf{w}^T \mathbf{w}^*$ by $\mathbf{x}_t^T \mathbf{w}^*$ (it could be negative), which is at least $1 - L_{hinge}(\mathbf{w}^*, \mathbf{x}_t)$. Similarly, on a negative example we "increase" $\mathbf{w}^T \mathbf{w}^*$ by $-\mathbf{x}^T_t \mathbf{w}^*$, which is also at least $1 - L_{hinge}(\mathbf{w}^*, \mathbf{x}_t)$. If we sum this up over all mistakes, we get that at the end we have $\mathbf{w}^T \mathbf{w}^* \geq M - L_{hinge}(\mathbf{w}^*, S)$, where we are using here the fact that hinge-loss is never negative so summing over all of S is only larger than summing over the mistakes that $\mathbf{w}$ made.

Finally, we just do some algebra. Let $L = L_{hinge}(\mathbf{w}^*, S)$. So we have:

$$\begin{aligned}
\mathbf{w}^T \mathbf{w}^* / |\mathbf{w}^*| &\leq |\mathbf{w}| \\
(\mathbf{w}_T \mathbf{w}_*)_2 &\leq |\mathbf{w}|^2 |\mathbf{w}^*|^2 \\
(M - L)^2 &\leq MR^2 |\mathbf{w}^*|^2 \\
M^2 - 2ML + L^2 &\leq MR^2 |\mathbf{w}^*|^2 \\
M - 2L + L^2/M &\leq R_2 |\mathbf{w}_*|_2 \\
M &\leq R^2 |\mathbf{w}^*|^2 + 2L - L^2/M \leq R^2 |\mathbf{w}^*|^2 + 2L
\end{aligned}$$

as desired. ∎

5.9 Online to Batch Conversion

Suppose we have an online algorithm with a good mistake bound, such as the Perceptron algorithm. Can we use it to get a guarantee in the distributional (batch) learning setting? Intuitively, the answer should be yes since the online setting is only harder. Indeed, this intuition is correct. We present here two natural approaches for such online to batch conversion.

Conversion procedure 1: Random Stopping. Suppose we have an online algorithm A with mistake-bound M. Say we run the algorithm in a single pass on a sample S of size $M/$. Let X_t be the indicator random variable for the event that A makes a mistake on example $\mathbf{x}_t$. Since $\sum_{t=1}^{|S|} X_t \leq M$ for *any* set S, we certainly have that $\mathbf{E}[\sum_{t=1}^{|S|} X_t] \leq M$ where the expectation is taken over the random draw of S from $D^{|S|}$. By linearity of expectation, and dividing both sides by $|S|$ we therefore have:

$$\frac{1}{|S|}\sum_{t=1}^{|S|}\mathbf{E}[X_t] \quad \leq \quad M/|S| = \epsilon. \tag{5.1}$$

Let h_t denote the hypothesis used by algorithm A to predict on the tth example. Since the tth example was randomly drawn from D, we have $\mathbf{E}[err_D(h_t)] = \mathbf{E}[X_t]$. This means that if we choose t at random from 1 to $|S|$, i.e., stop the algorithm at a random time, the expected error of the resulting prediction rule, taken over the randomness in the draw of S and the choice of t, is at most as given by equation (5.1). Thus we have:

Theorem 5.12 (Online to Batch via Random Stopping) *If an online algorithm* A *with mistake-bound M is run on a sample S of size M/ and stopped at a random time between* 1 *and* $|S|$*, the expected error of the hypothesis h produced satisfies* $\mathbf{E}[err_{\mathcal{D}}(h)] \leq \epsilon$.

Conversion procedure 2: Controlled Testing. A second natural approach to using an online learning algorithm A in the distributional setting is to just run a series of controlled tests. Specifically, suppose that the initial hypothesis produced by algorithm A is h_1. Define $\delta_i = \delta/(i+2)^2$ so we have $\sum_{i=0}^{\infty}\delta_i = (\frac{\pi^2}{6}-1)\delta \leq \delta$. We draw a set of $n_1 = \frac{1}{\epsilon}\log(\frac{1}{\delta_1})$ random examples and test to see whether h_1 gets all of them correct. Note that if $err_D(h_1) \geq \epsilon$ then the chance h_1 would get them all correct is at most $(1-\epsilon)^{n_1} \leq \delta_1$. So, if h_1 indeed gets them all correct, we output h_1 as our hypothesis and halt. If not, we choose some example x_1 in the sample on which h_1 made a mistake and give it to algorithm A. Algorithm A then produces some new hypothesis h_2 and we again repeat, testing h_2 on a fresh set of $n_2 = \frac{1}{\epsilon}\log(\frac{1}{\delta_2})$ random examples, and so on.

In general, given h_t we draw a fresh set of $n_t = \frac{1}{\epsilon}\log(\frac{1}{\delta_t})$ random examples and test to see whether h_t gets all of them correct. If so, we output h_t and halt; if not, we choose some x_t on which $h_t(x_t)$ was incorrect and give it to algorithm A. By choice of n_t, if h_t had error rate or larger, the chance we would mistakenly output it is at most δ_t. By choice of the values δ_t, the chance we *ever* halt with a hypothesis of error or larger is at most $\delta_1 + \delta_2 + \ldots \leq \delta$. Thus, we have the following theorem.

Theorem 5.13 (Online to Batch via Controlled Testing) *Let* A *be an online learning algorithm with mistake-bound M. Then this procedure will halt after* $O(\frac{M}{\epsilon}\log(\frac{M}{\delta}))$ *examples and with probability at least* $1 - \delta$ *will produce a hypothesis of error at most* .

Note that in this conversion we cannot re-use our samples: since the hypothesis h_t depends on the previous data, we need to draw a fresh set of n_t examples to use for testing it.

5.10 Support-Vector Machines

In a batch setting, rather than running the Perceptron algorithm and adapting it via one of the methods above, another natural idea would be just to solve for the vector $\mathbf{w}$ that minimizes the right-hand-side in Theorem 5.11 on the given dataset S. This turns out to have good guarantees as well, though they are beyond the scope of this book. In fact, this is the Support Vector Machine (SVM) algorithm. Specifically, SVMs solve the following convex optimization problem over a sample $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n\}$ where c is a constant that is determined empirically.

$$\begin{aligned} \text{minimize} \quad & c|\mathbf{w}|^2 + \sum_i s_i \\ \text{subject to} \quad & \mathbf{w} \cdot \mathbf{x}_i \geq 1 - s_i \text{ for all positive examples } \mathbf{x}_i \\ & \mathbf{w} \cdot \mathbf{x}_i \leq -1 + s_i \text{ for all negative examples } \mathbf{x}_i \\ & s_i \geq 0 \text{ for all } i. \end{aligned}$$

The variables s_i are called *slack variables*, and notice that the sum of the slack variables is the total hinge loss of $\mathbf{w}$. So, this convex optimization is minimizing a weighted sum of $1/\gamma^2$, where γ is the margin, and the total hinge loss. If we were to add the constraint that all $s_i = 0$ then this would be solving for the maximum margin linear separator for the data. However, in practice, optimizing a weighted combination generally performs better. SVMs can also be kernelized, by using the dual of the above optimization problem (the key idea is that the optimal $\mathbf{w}$ will be a weighted combination of data points, just as in the Perceptron algorithm, and these weights can be variables in the optimization problem); details are beyond the scope of this book.

5.11 VC-Dimension

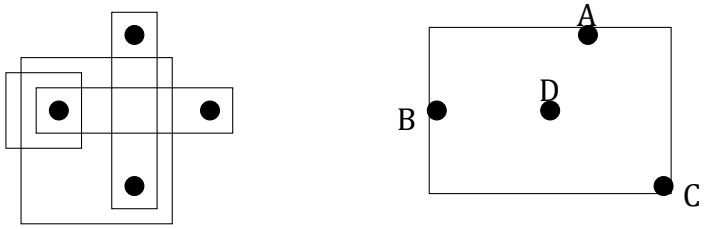
In Section 5.5 we presented several theorems showing that so long as the training set S is large compared to $\frac{1}{\epsilon}\log(|\mathcal{H}|)$, we can be confident that every $h \in H$ with $err_D(h) \geq \epsilon$ will have $err_S(h) > 0$, and if S is large compared to $\frac{1}{\epsilon^2}\log(|\mathcal{H}|)$, then we can be confident that every $h \in H$ will have $|err_D(h) - err_S(h)| \leq \epsilon$. In essence, these results used log(|H|) as a measure of complexity of class H. VC-dimension is a different, tighter measure of complexity for a concept class, and as we will see, is also sufficient to yield confidence bounds. For any class H, VCdim(H) ≤ $\log_2$(|H|) but it can also be quite a bit smaller. Let's introduce and motivate it through an example.

Consider a database consisting of the salary and age for a random sample of the adult population in the United States. Suppose we are interested in using the database to answer questions of the form: "what fraction of the adult population in the United States has age

between 35 and 45 and salary between $50,000 and $70,000?" That is, we are interested in queries that ask about the fraction of the adult population within some axisparallel rectangle. What we can do is calculate the fraction of the database satisfying this condition and return this as our answer. This brings up the following question: How large does our database need to be so that with probability greater than or equal to $1-\delta$, our answer will be within ± of the truth for *every* possible rectangle query of this form?

If we assume our values are discretized such as 100 possible ages and 1,000 possible salaries, then there are at most $(100\times1{,}000)^2 = 10^{10}$ possible rectangles. This means we can apply Theorem 5.5 with $|H| \leq 10^{10}$. Specifically, we can think of the target concept c^* as the empty set so that $err_S(h)$ is exactly the fraction of the sample inside rectangle h and $err_D(h)$ is exactly the fraction of the whole population inside h.[22] This would tell us that a sample size of $\frac{1}{2\epsilon^2}(10\ln 10 + \ln(2/\delta))$ would be sufficient.

However, what if we do not wish to discretize our concept class? Another approach would be to say that if there are only N adults total in the United States, then there are at most N^4 rectangles that are truly different with respect to D and so we could use $|H| \leq N^4$. Still, this suggests that S needs to grow with N, albeit logarithmically, and one might wonder if that is really necessary. VC-dimension, and the notion of the *growth function* of concept class H, will give us a way to avoid such discretization and avoid any dependence on the size of the support of the underlying distribution D.



(*a*) (*b*)

Figure 5.5: (a) shows a set of four points that can be shattered by rectangles along with some of the rectangles that shatter the set. Not every set of four points can be shattered as seen in (b). Any rectangle containing points A, B, and C must contain D. No set of five points can be shattered by rectangles with axis-parallel edges. No set of three collinear points can be shattered, since any rectangle that contains the two end points must also contain the middle point. More generally, since rectangles are convex, a set with one point inside the convex hull of the others cannot be shattered.

[22] Technically D is the uniform distribution over the adult population of the United States, and we want to think of S as an independent identically distributed sample from this D.

5.11.1 Definitions and Key Theorems

Definition 5.1 *Given a set S of examples and a concept class* H*, we say that S is* **shattered** *by* H *if for every* $A \subseteq S$ *there exists some* $h \in$ H *that labels all examples in A as positive and all examples in* $S \setminus A$ *as negative.*

Definition 5.2 *The* **VC-dimension** *of* H *is the size of the largest set shattered by* H*.*

For example, there exist sets of four points in the plane that can be shattered by rectangles with axis-parallel edges, e.g., four points at the vertices of a diamond (see Figure 5.5). Given such a set S, for any $A \subseteq S$, there exists a rectangle with the points in A inside the rectangle and the points in $S \setminus A$ outside the rectangle. However, rectangles with axis-parallel edges cannot shatter any set of five points. To see this, assume for contradiction that there is a set of five points shattered by the family of axis-parallel rectangles. Find the minimum enclosing rectangle for the five points. For each edge there is at least one point that has stopped its movement. Identify one such point for each edge. The same point may be identified as stopping two edges if it is at a corner of the minimum enclosing rectangle. If two or more points have stopped an edge, designate only one as having stopped the edge. Now, at most four points have been designated. Any rectangle enclosing the designated points must include the undesignated points. Thus, the subset of designated points cannot be expressed as the intersection of a rectangle with the five points. Therefore, the VC-dimension of axis-parallel rectangles is four.

We now need one more definition, which is the *growth function* of a concept class H.

Definition 5.3 *Given a set S of examples and a concept class* H*, let* $\text{H}[S] = \{h \cap S : h \in \text{H}\}$*. That is,* H[$S$] *is the concept class* H *restricted to the set of points S. For integer n and class* H*, let* $\text{H}[n] = \max_{|S|=n} |\text{H}[S]|$*; this is called the* **growth function** *of* H*.*

For example, we could have defined shattering by saying that S is shattered by H if $|\text{H}[S]| = 2^{|S|}$, and then the VC-dimension of H is the largest n such that $\text{H}[n] = 2^n$. Notice also that for axis-parallel rectangles, $\text{H}[n] = O(n^4)$. The growth function of a class is sometimes called the shatter function or shatter coefficient.

What connects these to learnability are the following three remarkable theorems. The first two are analogs of Theorem 5.3 and Theorem 5.5 respectively, showing that one can replace |H| with its growth function. This is like replacing the number of concepts in H with the number of concepts "after the fact", i.e., after S is drawn, and is subtle because we cannot just use a union bound after we have already drawn our set S. The third theorem relates the growth function of a class to its VC-dimension. We now present the theorems, give examples of VC-dimension and growth function of various concept classes, and then prove the theorems.

Theorem 5.14 (Growth function sample bound) *For any class* H *and distribution* D*, if a training sample S is drawn from* D *of size*

$$n \geq \frac{2}{\epsilon}[\log_2(2\mathcal{H}[2n]) + \log_2(1/\delta)]$$

then with probability ≥ 1−*δ, every h* ∈ H *with* $err_{\mathcal{D}}(h) \geq \epsilon$ *has* $err_S(h) > 0$ *(equivalently, every h* ∈ H *with* $err_S(h) = 0$ *has* $err_{\mathcal{D}}(h) < \epsilon$*).*

Theorem 5.15 (Growth function uniform convergence) *For any class* H *and distribution* D*, if a training sample S is drawn from* D *of size*

$$n \geq \frac{8}{\epsilon^2}[\ln(2\mathcal{H}[2n]) + \ln(1/\delta)]$$

then with probability ≥ 1 − *δ, every h* ∈ H *will have* $|err_S(h) - err_{\mathcal{D}}(h)| \leq \epsilon$*.*

Theorem 5.16 (Sauer's lemma) *If* VCdim(H) = *d then* $\mathcal{H}[n] \leq \sum_{i=0}^{d} \binom{n}{i} \leq (\frac{en}{d})^d$*.*

Notice that Sauer's lemma was fairly tight in the case of axis-parallel rectangles, though in some cases it can be a bit loose. E.g., we will see that for linear separators in the plane, their VC-dimension is 3 but H[n] = $O(n^2)$. An interesting feature about Sauer's lemma is that it implies the growth function switches from taking the form 2^n to taking the form of roughly $n^{\text{VCdim(H)}}$ when n exceeds the VC-dimension of the class H.

Putting Theorems 5.14 and 5.16 together, with a little algebra we get the following corollary (a similar corollary results by combining Theorems 5.15 and 5.16):

Corollary 5.17 (VC-dimension sample bound) *For any class* H *and distribution* D*, a training sample S of size*

$$O\left(\frac{1}{\epsilon}[\text{VCdim}(\mathcal{H})\log(1/\epsilon) + \log(1/\delta)]\right)$$

is sufficient to ensure that with probability ≥ 1 − *δ, every h* ∈ H *with* $err_{\mathcal{D}}(h) \geq \epsilon$ *has* $err_S(h) > 0$ *(equivalently, every h* ∈ H *with* $err_S(h) = 0$ *has* $err_{\mathcal{D}}(h) < \epsilon$*).*

For any class H, VCdim(H) ≤ $\log_2(|H|)$ since H must have at least 2^k concepts in order to shatter k points. Thus Corollary 5.17 is never too much worse than Theorem 5.3 and can be much better.

5.11.2 Examples: VC-Dimension and Growth Function

Rectangles with axis-parallel edges

As we saw above, the class of axis-parallel rectangles in the plane has VC-dimension 4 and growth function H[n] = $O(n^4)$.

Intervals of the reals

Intervals on the real line can shatter any set of two points but no set of three points since the subset of the first and last points cannot be isolated. Thus, the VC-dimension of intervals is two. Also, $H[n] = O(n^2)$ since we have $O(n^2)$ choices for the left and right endpoints.

Pairs of intervals of the reals

Consider the family of pairs of intervals, where a pair of intervals is viewed as the set of points that are in at least one of the intervals, in other words, their set union. There exists a set of size four that can be shattered but no set of size five since the subset of first, third, and last point cannot be isolated. Thus, the VC-dimension of pairs of intervals is four. Also we have $H[n] = O(n^4)$.

Convex polygons

Consider the set system of all convex polygons in the plane. For any positive integer n, place n points on the unit circle. Any subset of the points are the vertices of a convex polygon. Clearly that polygon will not contain any of the points not in the subset. This shows that convex polygons can shatter arbitrarily large sets, so the VC-dimension is infinite. Notice that this also implies that $H[n] = 2^n$.

Halfspaces in d-dimensions

Define a halfspace to be the set of all points on one side of a linear separator, i.e., a set of the form $\{\mathbf{x}|\mathbf{w}^T\mathbf{x} \geq w_0\}$. The VC-dimension of halfspaces in d-dimensions is d+1.

There exists a set of size $d + 1$ that can be shattered by halfspaces. Select the d unitcoordinate vectors plus the origin to be the d+1 points. Suppose A is any subset of these d+1 points. Without loss of generality assume that the origin is in A. Take a 0-1 vector $\mathbf{w}$ which has 1's precisely in the coordinates corresponding to vectors not in A. Clearly A lies in the half-space $\mathbf{w}^T\mathbf{x} \leq 0$ and the complement of A lies in the complementary halfspace.

We now show that no set of $d + 2$ points in d-dimensions can be shattered by halfspaces. This is done by proving that any set of $d + 2$ points can be partitioned into two disjoint subsets A and B of points whose convex hulls intersect. This establishes the claim since any linear separator with A on one side must have its entire convex hull on that

side,[23] so it is not possible to have a linear separator with A on one side and B on the other. Let $convex(S)$ denote the convex hull of point set S.

Theorem 5.18 (**Radon**)*: Any set $S \subseteq R^d$ with $|S| \geq d + 2$, can be partitioned into two disjoint subsets A and B such that $convex(A) \cap convex(B) \neq \varphi$.*

Proof: Without loss of generality, assume $|S| = d+2$. Form a $d \times (d+2)$ matrix with one column for each point of S. Call the matrix A. Add an extra row of all 1's to construct a $(d+1) \times (d+2)$ matrix B. Clearly the rank of this matrix is at most $d+1$ and the columns are linearly dependent. Say $\mathbf{x} = (x_1, x_2, \ldots, x_{d+2})$ is a nonzero vector with $B\mathbf{x} = 0$. Reorder the columns so that $x_1, x_2, \ldots, x_s \geq 0$ and $x_{s+1}, x_{s+2}, \ldots, x_{d+2} < 0$. Normalize $\mathbf{x}$ so $\sum_{i=1}^{s} |x_i| = 1$. Let $\mathbf{b_i}$ (respectively $\mathbf{a_i}$) be the i^{th} column of B (respectively A). Then,

$$\sum_{i=1}^{s} |x_i|\mathbf{b_i} = \sum_{i=s+1}^{d+2} |x_i|\mathbf{b_i}$$ from which it follows that $$\sum_{i=1}^{s} |x_i|\mathbf{a_i} = \sum_{i=s+1}^{d+2} |x_i|\mathbf{a_i}$$ and $$\sum_{i=1}^{s} |x_i| = \sum_{i=s+1}^{d+2} |x_i|.$$ Since $\sum_{i=1}^{s} |x_i| = 1$ and $\sum_{i=s+1}^{d+2} |x_i| = 1$ each side of $\sum_{i=1}^{s} |x_i|\mathbf{a_i} = \sum_{i=s+1}^{d+2} |x_i|\mathbf{a_i}$ is a convex combination of columns of A which proves the theorem. Thus, S can be partitioned into two sets, the first consisting of the first s points after the rearrangement and the second consisting of points $s + 1$ through $d + 2$. Their convex hulls intersect as required. ■

Radon's theorem immediately implies that half-spaces in d-dimensions do not shatter any set of $d + 2$ points.

Spheres in d-dimensions

A *sphere* in d-dimensions is a set of points of the form $\{\mathbf{x} \mid |\mathbf{x} - \mathbf{x_0}| \leq r\}$. The VCdimension of spheres is $d+1$. It is the same as that of halfspaces. First, we prove that no set of $d+2$ points can be shattered by spheres. Suppose some set S with $d+2$ points can be shattered. Then for any partition A_1 and A_2 of S, there are spheres B_1 and B_2 such that $B_1 \cap S = A_1$ and $B_2 \cap S = A_2$. Now B_1 and B_2 may intersect, but there is no point of S in their intersection. It is easy to see that there is a hyperplane perpendicular to the line joining the centers of the two spheres with all of A_1 on one side and all of A_2 on the other and this implies that

[23] If any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ lie on the same side of a linear separator, so must any convex combination: if $\mathbf{w} \cdot \mathbf{x}_1 \geq b$ and $\mathbf{w} \cdot \mathbf{x}_2 \geq b$ then $\mathbf{w} \cdot (a\mathbf{x}_1 + (1 - a)\mathbf{x}_2) \geq b$.

halfspaces shatter *S*, a contradiction. Therefore no $d + 2$ points can be shattered by hyperspheres.

It is also not difficult to see that the set of $d+1$ points consisting of the unit-coordinate vectors and the origin can be shattered by spheres. Suppose *A* is a subset of the $d + 1$ points. Let *a* be the number of unit vectors in *A.* The center $\mathbf{a_0}$ of our sphere will be the sum of the vectors in *A*. For every unit vector in *A*, its distance to this center will be $\sqrt{a - 1}$ and for every unit vector outside *A*, its distance to this center will be $\sqrt{a + 1}$. The distance of the origin to the center is *a*. Thus, we can choose the radius so that precisely the points in *A* are in the hypersphere.

Finite sets

The system of finite sets of real numbers can shatter any finite set of real numbers and thus the VC-dimension of finite sets is infinite.

5.11.3 Proof of Main Theorems

We begin with a technical lemma. Consider drawing a set *S* of *n* examples from D and let *A* denote the event that there exists $h \in$ H with zero training error on *S* but true error greater than or equal to . Now draw a second set S^0 of *n* examples from D and let *B* denote the event that there exists $h \in$ H with zero error on *S* but error greater than or equal to $\epsilon/2$ on S'. We claim that Prob(*B*) ≥ Prob(*A*)/2.

Lemma 5.19 *Let* H *be a concept class over some domain* X *and let S and* S^0 *be sets of n elements drawn from some distribution* D *on* X*, where* $n \geq 8/\epsilon$*. Let A be the event that there exists* $h \in$ H *with zero error on S but true error greater than or equal to . Let B be the event that there exists* $h \in$ H *with zero error on S but error greater than or equal to* $\frac{\epsilon}{2}$ *on* S^0*. Then Prob*(*B*) ≥ *Prob*(*A*)/2*.*

Proof: Clearly, Prob(*B*) ≥ Prob(*A,B*) = Prob(*A*)Prob(*B*|*A*). Consider drawing set *S* and suppose event *A* occurs. Let *h* be in H with $err_{\mathcal{D}}(h) \geq \epsilon$ but $err_S(h) = 0$. Now, draw set S^0. **E**(error of $h \text{ on } S') = err_{\mathcal{D}}(h) \geq \epsilon$. So, by Chernoff bounds, since $n \geq 8/\epsilon$, Prob($err_{S'}(h) \geq \epsilon/2) \geq 1/$ 2. Thus, Prob(*B*|*A*) ≥ 1/2 and Prob(*B*) ≥ Prob(*A*)/2 as desired. ■

We now prove Theorem 5.14, restated here for convenience.

Theorem 5.14 (Growth function sample bound) *For any class* H *and distribution* D*, if a training sample S is drawn from* D *of size*

$$n \geq \frac{2}{\epsilon}[\log_2(2\mathcal{H}[2n]) + \log_2(1/\delta)]$$

then with probability ≥ 1−*δ, every h* ∈ H *with* $err_{\mathcal{D}}(h) \geq \epsilon$ *has* $err_S(h) > 0$ *(equivalently, every h* ∈ H *with* $err_S(h) = 0$ *has* $err_{\mathcal{D}}(h) < \epsilon$*).*

Proof: Consider drawing a set *S* of *n* examples from D and let *A* denote the event that there exists *h* ∈ H with true error greater than but training error zero. Our goal is to prove that Prob(*A*) ≤ *δ*.

By Lemma 5.19 it suffices to prove that Prob(*B*) ≤ *δ*/2. Consider a third experiment. Draw a set S^{00} of 2*n* points from D and then randomly partition S^{00} into two sets *S* and S^0 of *n* points each. Let B^* denote the event that there exists *h* ∈ H with $err_S(h) = 0$ but $err_{S'}(h) \geq \epsilon/2$. Prob($B^*$) = Prob(*B*) since drawing 2*n* points from D and randomly partitioning them into two sets of size *n* produces the same distribution on (S,S^0) as does drawing *S* and S^0 directly. The advantage of this new experiment is that we can now argue that Prob(B^*) is low by arguing that for any set S^{00} of size 2*n*, Prob($B^*|S^{00}$) is low, with probability now taken over just the random partition of S^{00} into *S* and S^0. The key point is that since S^{00} is fixed, there are at most $|H[S^{00}]| \leq H[2n]$ events to worry about. Specifically, it suffices to prove that for any fixed *h* ∈ H[S^{00}], the probability over the partition of S^{00} that *h* makes zero mistakes on *S* but more than *n*/2 mistakes on S^0 is at most *δ*/(2H[2*n*]). We can then apply the union bound over H[S^{00}] = {*h* ∩ S^{00}|*h* ∈ H}.

To make the calculations easier, consider the following specific method for partitioning S^{00} into *S* and S^0. Randomly put the points in S^{00} into pairs: (a_1,b_1), (a_2,b_2), ..., (a_n,b_n). For each index *i*, flip a fair coin. If heads put a_i into *S* and b_i into S^0, else if tails put a_i into S^0 and b_i into *S*. Now, fix some partition *h* ∈ H[S^{00}] and consider the probability over these *n* fair coin flips that *h* makes zero mistakes on *S* but more than *n*/2 mistakes on S^0. First of all, if for any index *i*, *h* makes a mistake on both a_i and b_i then the probability is zero (because it cannot possibly make zero mistakes on *S*). Second, if there are fewer than *n*/2 indices *i* such that *h* makes a mistake on either a_i or b_i then again the probability is zero because it cannot possibly make more than *n*/2 mistakes on S^0. So, assume there are $r \geq \epsilon n/2$ indices *i* such that *h* makes a mistake on exactly one of a_i or b_i. In this case, the chance that all of those mistakes land in S^0 is exactly $1/2^r$. This quantity is at most $1/2^{\epsilon n/2} \leq \delta/(2\mathcal{H}[2n])$ as desired for *n* as given in the theorem statement. ■

We now prove Theorem 5.15, restated here for convenience.

Theorem 5.15 (Growth function uniform convergence) *For any class* H *and distribution* D*, if a training sample S is drawn from* D *of size*

$$n \geq \frac{8}{\epsilon^2}[\ln(2\mathcal{H}[2n]) + \ln(1/\delta)]$$

then with probability ≥ 1 − *δ, every h* ∈ H *will have* $|err_S(h) - err_{\mathcal{D}}(h)| \leq \epsilon$.

Proof: This proof is identical to the proof of Theorem 5.14 except B^* is now the event that there exists a set $h \in \mathrm{H}[S^{00}]$ such that the error of h on S differs from the error of h on S^0 by more than $/2$. We again consider the experiment where we randomly put the points in S^{00} into pairs (a_i,b_i) and then flip a fair coin for each index i, if heads placing a_i into S and b_i into S^0, else placing a_i into S^0 and b_i into S. Consider the difference between the number of mistakes h makes on S and the number of mistakes h makes on S^0 and observe how this difference changes as we flip coins for $i = 1,2,\ldots,n$. Initially, the difference is zero. If h makes a mistake on both or neither of (a_i,b_i) then the difference does not change. Else, if h makes a mistake on exactly one of a_i or b_i, then with probability $1/2$ the difference increases by one and with probability $1/2$ the difference decreases by one. If there are $r \le n$ such pairs, then if we take a random walk of $r \le n$ steps, what is the probability that we end up more than $n/2$ steps away from the origin? This is equivalent to asking: if we flip $r \le n$ fair coins, what is the probability the number of heads differs from its expectation by more than $n/4$. By Hoeffding bounds, this is at most $2e^{-\epsilon^2 n/8}$. This quantity is at most $\delta/(2\mathrm{H}[2n])$ as desired for n as given in the theorem statement.

■

Finally, we prove Sauer's lemma, relating the growth function to the VC-dimension.

Theorem 5.16 (Sauer's lemma) *If* VCdim(H) = d *then* $\mathcal{H}[n] \le \sum_{i=0}^{d} \binom{n}{i} \le (\frac{en}{d})^d$.

Proof: Let d = VCdim(H). Our goal is to prove for any set S of n points that $|\mathcal{H}[S]| \le \binom{n}{\le d}$, where we are defining $\binom{n}{\le d} = \sum_{i=0}^{d} \binom{n}{i}$; this is the number of distinct ways of choosing d or fewer elements out of n. We will do so by induction on n. As a base case, our theorem is trivially true if $n \le d$.

As a first step in the proof, notice that:

$$\binom{n}{\le d} = \binom{n-1}{\le d} + \binom{n-1}{\le d-1} \tag{5.2}$$

because we can partition the ways of choosing d or fewer items into those that do not include the first item (leaving $\le d$ to be chosen from the remainder) and those that do include the first item (leaving $\le d - 1$ to be chosen from the remainder).

Now, consider any set S of n points and pick some arbitrary point $x \in S$. By induction, we may assume that $|\mathcal{H}[S \setminus \{x\}]| \le \binom{n-1}{\le d}$. So, by equation (5.2) all we need to show is that $|\mathcal{H}[S]| - |\mathcal{H}[S \setminus \{x\}]| \le \binom{n-1}{\le d-1}$. Thus, our problem has reduced to analyzing how many *more* partitions there are of S than there are of $S \setminus \{x\}$ using sets in H.

If H[S] is larger than H[$S \setminus \{x\}$], it is because of pairs of sets in H[S] that differ only on point x and therefore collapse to the same set when x is removed. For set $h \in$ H[S] containing point x, define twin(h) = $h \setminus \{x\}$; this may or may not belong to H[S]. Let T = {$h \in$ H[S] : $x \in h$ and twin(h) $\in$ H[S]}. Notice |H[S]| − |H[$S \setminus \{x\}$]| = |T |.

Now, what is the VC-dimension of T ? If d^0 = VCdim(T), this means there is some set R of d^0 points in $S \setminus \{x\}$ that are shattered by T . By definition of T , all 2^{d_0} subsets of R can be extended to either include x, or not include x and still be a set in H[S]. In other words, $R \cup \{x\}$ is shattered by H. This means, $d^0 + 1 \le d$. Since VCdim(T) $\le d - 1$, by induction we have $|\mathcal{T}| \le \binom{n-1}{\le d-1}$ as desired. ■

5.11.4 VC-Dimension of Combinations of Concepts

Often one wants to create concepts out of other concepts. For example, given several linear separators, one could take their intersection to create a convex polytope. Or given several disjunctions, one might want to take their majority vote. We can use Sauer's lemma to show that such combinations do not increase the VC-dimension of the class by too much.

Specifically, given k concepts $h_1,h_2,...,h_k$ and a Booelan function f define the set $comb_f(h_1,...,h_k) = \{x \in X : f(h_1(x),...,h_k(x)) = 1\}$, where here we are using $h_i(x)$ to denote the indicator for whether or not $x \in h_i$. For example, f might be the AND function to take the intersection of the sets h_i, or f might be the majority-vote function. This can be viewed as a *depth-two neural network*. Given a concept class H, a Boolean function f, and an integer k, define the new concept class $COMB_{f,k}$(H) = {$comb_f(h_1,...,h_k) : h_i \in$ H}. We can now use Sauer's lemma to produce the following corollary.

Corollary 5.20 *If the concept class* H *has VC-dimension d, then for any combination function f, the class* $COMB_{f,k}$(H) *has VC-dimension* $O\big(kd\log(kd)\big)$.

Proof: Let n be the VC-dimension of $\text{COMB}_{f,k}$(H), so by definition, there must exist a set S of n points shattered by $\text{COMB}_{f,k}$(H). We know by Sauer's lemma that there are at most n^d ways of partitioning the points in S using sets in H. Since each set in $\text{COMB}_{f,k}$(H) is determined by k sets in H, and there are at most $(n^d)^k = n^{kd}$ different k-tuples of such sets, this means there are at most n^{kd} ways of partitioning the points using sets in $\text{COMB}_{f,k}$(H). Since S is shattered, we must have $2^n \le n^{kd}$, or equivalently√

—

$kdn \le \log kd_2(\log n)_2 \le (nkd)$. We solve this as follows. First, assuming√n which implies that $n \le (kd)_2$. To get the better bound, plug back into$n \ge 16$ we have $\log_2(n) \le$ n so

the original inequality. Since $n \le (kd)^2$, it must be that $\log_2(n) \le 2\log_2(kd)$. Substituting $\log n \le 2\log_2(kd)$ into $n \le kd\log_2 n$ gives $n \le 2kd\log_2(kd)$. ■

This result will be useful for our discussion of Boosting in Section 5.12.

5.11.5 Other Measures of Complexity

VC-dimension and number of bits needed to describe a set are not the only measures of complexity one can use to derive generalization guarantees. There has been significant work on a variety of measures. One measure called Rademacher complexity measures the extent to which a given concept class H can fit random noise. Given a set of n examples $S = \{x_1,...,x_n\}$, the *empirical Rademacher complexity* of H is defined as $R_S(\mathcal{H}) = \mathbf{E}_{\sigma_1,...,\sigma_n} \max_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i h(x_i)$, where $\sigma_i \in \{-1,1\}$ are independent random labels with Prob$[\sigma_i = 1] = \frac{1}{2}$. E.g., if you assign random ±1 labels to the points in S and the best classifier in H on average gets error 0.45 then $R_S(H) = 0.55 - 0.45 = 0.1$. One can prove that with probability greater than or equal to $1-\delta$, every $h \in H$ satisfies true error less than or equal to training error plus $R_S(\mathcal{H}) + 3\sqrt{\frac{\ln(2/\delta)}{2n}}$. For more on results such as this, see, e.g., [BM02].

5.12 Strong and Weak Learning - Boosting

We now describe *boosting*, which is important both as a theoretical result and as a practical and easy-to-use learning method.

A *strong learner* for a problem is an algorithm that with high probability is able to achieve any desired error rate using a number of samples that may depend polynomially on 1/. A *weak learner* for a problem is an algorithm that does just a little bit better than random guessing. It is only required to get with high probability an error rate less than or equal to $\frac{1}{2} - \gamma$ for some $0 < \gamma \le \frac{1}{2}$. We show here that a weak-learner for a problem that achieves the weak-learning guarantee for any distribution of data can be boosted to a strong learner, using the technique of boosting. At the high level, the idea will be to take our training sample S, and then to run the weak-learner on different data distributions produced by weighting the points in the training sample in different ways. Running the weak learner on these different weightings of the training sample will produce a series of hypotheses $h_1,h_2,...$, and the idea of our reweighting procedure will be to focus attention on the parts of the sample that previous hypotheses have performed poorly on. At the end we will combine the hypotheses together by a majority vote.

Assume the weak learning algorithm A outputs hypotheses from some class H. Our boosting algorithm will produce hypotheses that will be majority votes over t_0 hypotheses

from H, for t_0 defined below. This means that we can apply Corollary 5.20 to bound the VC-dimension of the class of hypotheses our boosting algorithm can produce in terms of the VC-dimension of H. In particular, the class of rules that can be produced by the booster running for t_0 rounds has VC-dimension $O(t_0\text{VCdim(H)}\log(t_0\text{VCdim(H)}))$. This in turn gives a bound on the number of samples needed, via Corollary 5.17, to ensure that high accuracy on the sample will translate to high accuracy on new data.

To make the discussion simpler, we will assume that the weak learning algorithm A, when presented with a weighting of the points in our training sample, always (rather than with high probability) produces a hypothesis that performs slightly better than random guessing with respect to the distribution induced by weighting. Specifically:

Boosting Algorithm

Given a sample S of n labeled examples $\mathbf{x}_1,\ldots,\mathbf{x}_n$, initialize each example $\mathbf{x}_i$ to have a weight $w_i = 1$. Let $\mathbf{w} = (w_1,\ldots,w_n)$.

For $t = 1,2,\ldots,t_0$ do

Call the weak learner on the weighted sample $(S,\mathbf{w})$, receiving hypothesis h_t.

Multiply the weight of each example that was misclassified by h_t by $\alpha = \frac{\frac{1}{2}+\gamma}{\frac{1}{2}-\gamma}$. Leave the other weights as they are.

End

Output the classifier MAJ$(h_1,\ldots,h_{t_0})$ which takes the majority vote of the hypotheses returned by the weak learner. Assume t_0 is odd so there is no tie.

Figure 5.6: The boosting algorithm

Definition 5.4 (γ-Weak learner on sample) *A γ-weak learner is an algorithm that given examples, their labels, and a nonnegative real weight w_i on each example $\mathbf{x}_i$, produces a classifier that correctly labels a subset of examples with total weight at least* $(\frac{1}{2}+\gamma)\sum_{i=1}^{n} w_i$.

At the high level, boosting makes use of the intuitive notion that if an example was misclassified, one needs to pay more attention to it. The boosting procedure is in Figure 5.6.

Theorem 5.21 *Let A be a γ-weak learner for sample S. Then* $t_0 = O(\frac{1}{\gamma^2}\log n)$ *is sufficient so that the classifier MAJ*(h_1,...,h_{t_0}) *produced by the boosting procedure has training error zero.*

Proof: Suppose m is the number of examples the final classifier gets wrong. Each of these m examples was misclassified at least $t_0/2$ times so each has weight at least $\alpha^{t_0/2}$. Thus the total weight is at least $m\alpha^{t_0/2}$. On the other hand, at time t+1, only the weights of examples misclassified at time t were increased. By the property of weak learning, the total weight of misclassified examples is at most $(\frac{1}{2} - \gamma)$ of the total weight at time t. Let weight(t) be the total weight at time t. Then

$$\text{weight}(t+1) \leq \left(\alpha\left(\tfrac{1}{2}-\gamma\right) + \left(\tfrac{1}{2}+\gamma\right)\right) \times \text{weight}(t)$$
$$= (1 + 2\gamma) \times \text{weight}(t).$$

Since weight(0) = n, the total weight at the end is at most $n(1 + 2\gamma)^{t_0}$. Thus

$$m\alpha^{t_0/2} \leq \text{total weight at end} \leq n(1 + 2\gamma)^{t_0}.$$

Substituting $\alpha = \frac{1/2+\gamma}{1/2-\gamma} = \frac{1+2\gamma}{1-2\gamma}$ and rearranging terms

$$m \leq n(1 - 2\gamma)^{t_0/2}(1 + 2\gamma)^{t_0/2} = n[1 - 4\gamma^2]^{t_0/2}.$$

Using $1 - x \leq e^{-x}$, $m \leq ne^{-2t_0\gamma^2}$. For $t_0 > \frac{\ln n}{2\gamma^2}$, $m < 1$, so the number of misclassified items must be zero. ■

Having completed the proof of the boosting result, here are two interesting observations:

Connection to Hoeffding bounds: The boosting result applies even if our weak learning algorithm is "adversarial", giving us the least helpful classifier possible subject to Definition 5.4. This is why we don't want the α in the boosting algorithm to be too large, otherwise the weak learner could return the negation of the classifier it gave the last time. Suppose that the weak learning algorithm gave a classifier each time that for each example, flipped a coin and produced the correct answer with probability $\frac{1}{2} + \gamma$ and the wrong answer with probability $\frac{1}{2} - \gamma$, so it is a γ-weak learner in expectation. In that case, if we called the weak learner t_0 times, for any fixed $\mathbf{x}_i$, Hoeffding bounds imply the chance the majority vote of those classifiers is incorrect on $\mathbf{x}_i$ is at most $e^{-2t_0\gamma^2}$. So, the expected total number of mistakes m is at most $ne^{-2t_0\gamma^2}$. What is interesting is that this is the exact bound we get from boosting without the expectation for an adversarial weak-learner.

A minimax view: Consider a 2-player zero-sum game [24] with one row for each example $\mathbf{x}_i$ and one column for each hypothesis h_j that the weak-learning algorithm might output. If the row player chooses row i and the column player chooses column j, then the column player gets a payoff of one if $h_j(\mathbf{x}_i)$ is correct and gets a payoff of zero if $h_j(\mathbf{x}_i)$ is incorrect. The γ-weak learning assumption implies that for any randomized strategy for the row player (any "mixed strategy" in the language of game theory), there exists a response h_j that gives the column player an expected payoff of at least $\frac{1}{2} + \gamma$. The von Neumann minimax theorem [25] states that this implies there exists a probability distribution on the columns (a mixed strategy for the column player) such that for any $\mathbf{x}_i$, at least a $\frac{1}{2} + \gamma$ probability mass of the columns under this distribution is correct on $\mathbf{x}_i$. We can think of boosting as a fast way of finding a very simple probability distribution on the columns (just an average over $O(\log n)$ columns, possibly with repetitions) that is nearly as good (for any $\mathbf{x}_i$, more than half are correct) that moreover works even if our only access to the columns is by running the weak learner and observing its outputs.

We argued above that $t_0 = O(\frac{1}{\gamma^2} \log n)$ rounds of boosting are sufficient to produce a majority-vote rule h that will classify all of S correctly. Using our VC-dimension bounds, this implies that if the weak learner is choosing its hypotheses from concept class H, then a sample size

$$n = \tilde{O}\left(\frac{1}{\epsilon}\left(\frac{\mathrm{VCdim}(\mathcal{H})}{\gamma^2}\right)\right)$$

is sufficient to conclude that with probability $1 - \delta$ the error is less than or equal to , where we are using the $\tilde{O}$ notation to hide logarithmic factors. It turns out that running the boosting procedure for larger values of t_0 i.e., continuing past the point where S is classified correctly by the final majority vote, does not actually lead to greater overfitting. The reason is that using the same type of analysis used to prove Theorem 5.21, one can show that as t_0 increases, not only will the majority vote be correct on each $\mathbf{x} \in S$, but in fact each example will be correctly classified by a $\frac{1}{2} + \gamma'$ fraction of the classifiers, where $\gamma^0 \to \gamma$ as $t_0 \to \infty$. I.e., the vote is approaching the minimax optimal strategy for the column player in the minimax view given above. This in turn implies that h can be well-approximated over S by a vote of a random sample of $O(1/\gamma^2)$ of its component weak

[24] A two person zero sum game consists of a matrix whose columns correspond to moves for Player 1 and whose rows correspond to moves for Player 2. The ij^{th} entry of the matrix is the payoff for Player 1 if Player 1 choose the j^{th} column and Player 2 choose the i^{th} row. Player 2's payoff is the negative of Player1's.

[25] The von Neumann minimax theorem states that there exists a mixed strategy for each player so that given Player 2's strategy the best payoff possible for Player 1 is the negative of given Player 1's strategy the best possible payoff for Player 2. A mixed strategy is one in which a probability is assigned to every possible move for each situation a player could be in.

hypotheses h_j. Since these small random majority votes are not overfitting by much, our generalization theorems imply that h cannot be overfitting by much either.

5.13 Stochastic Gradient Descent

We now describe a widely-used algorithm in machine learning, called *stochastic gradient descent* (SGD). The Perceptron algorithm we examined in Section 5.8.3 can be viewed as a special case of this algorithm, as can methods for deep learning.

Let F be a class of real-valued functions $f_{\mathbf{w}} : \mathsf{R}^d \to \mathsf{R}$ where $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ is a vector of parameters. For example, we could think of the class of linear functions where $n = d$ and $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T x$, or we could have more complicated functions where $n > d$. For each such function $f_{\mathbf{w}}$ we can define an associated set $h_{\mathbf{w}} = \{\mathbf{x} : f_{\mathbf{w}}(\mathbf{x}) \geq 0\}$, and let $\mathrm{H_F} = \{h_{\mathbf{w}} : f_{\mathbf{w}} \in \mathrm{F}\}$. For example, if F is the class of linear functions then $\mathrm{H_F}$ is the class of linear separators.

To apply stochastic gradient descent, we also need a *loss function* $L(f_{\mathbf{w}}(\mathbf{x}), c^*(\mathbf{x}))$ that describes the real-valued penalty we will associate with function $f_{\mathbf{w}}$ for its prediction on an example $\mathbf{x}$ whose true label is $c^*(\mathbf{x})$. The algorithm is then the following:

Stochastic Gradient Descent:

Given: starting point $\mathbf{w} = \mathbf{w}_{init}$ and learning rates $\lambda_1, \lambda_2, \lambda_3, \ldots$

(e.g., $\mathbf{w}_{init} = \mathbf{0}$ and $\lambda_t = 1$ for all t, or $\lambda_t = 1/\sqrt{t}$).

Consider a sequence of random examples $(\mathbf{x}_1, c^*(\mathbf{x}_1)), (\mathbf{x}_2, c^*(\mathbf{x}_2)), \ldots$.

1. Given example $(\mathbf{x}_t, c^*(\mathbf{x}_t))$, compute the gradient $\nabla L(f_{\mathbf{w}}(\mathbf{x}_t), c^*(\mathbf{x}_t))$ of the loss of $f_{\mathbf{w}}(\mathbf{x}_t)$ with respect to the weights $\mathbf{w}$. This is a vector in R^n whose ith component is $\frac{\partial L(f_{\mathbf{w}}(\mathbf{x}_t), c^*(\mathbf{x}_t))}{\partial w_i}$.
2. Update: $\mathbf{w} \leftarrow \mathbf{w} - \lambda_t \nabla L(f_{\mathbf{w}}(\mathbf{x}_t), c^*(\mathbf{x}_t))$.

Let's now try to understand the algorithm better by seeing a few examples of instantiating the class of functions F and loss function L.

First, consider $n = d$ and $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$, so F is the class of linear predictors. Consider the loss function $L(f_{\mathbf{w}}(\mathbf{x}), c^*(\mathbf{x})) = \max(0, -c^*(\mathbf{x}) f_{\mathbf{w}}(\mathbf{x}))$, and recall that $c^*(\mathbf{x}) \in \{-1, 1\}$. In other words, if $f_{\mathbf{w}}(\mathbf{x})$ has the correct sign, then we have a loss of 0, otherwise we have a loss equal to the magnitude of $f_{\mathbf{w}}(\mathbf{x})$. In this case, if $f_{\mathbf{w}}(\mathbf{x})$ has the correct sign and is non-zero, then the gradient will be zero since an infinitesimal change in any of the weights will not change the sign. So, when $h_{\mathbf{w}}(\mathbf{x})$ is correct, the algorithm will leave $\mathbf{w}$ alone.

On the other hand, if $f_{\mathbf{w}}(\mathbf{x})$ has the wrong sign, then $\frac{\partial L}{\partial w_i} = -c^*(\mathbf{x})\frac{\partial \mathbf{w}\cdot\mathbf{x}}{\partial w_i} = -c^*(\mathbf{x})x_i$. So, using $\lambda_t = 1$, the algorithm will update $w \leftarrow w + c^*(\mathbf{x})\mathbf{x}$. Note that this is exactly the Perceptron algorithm. (Technically we must address the case that $f_{\mathbf{w}}(\mathbf{x}) = 0$; in this case, we should view $f_{\mathbf{w}}$ as having the wrong sign just barely.)

As a small modification to the above example, consider the same class of linear predictors F but now modify the loss function to the hinge-loss $L(f_{\mathbf{w}}(\mathbf{x}),c^*(\mathbf{x})) = \max(0,1 - c^*(\mathbf{x})f_{\mathbf{w}}(\mathbf{x}))$. This loss function now requires $f_{\mathbf{w}}(\mathbf{x})$ to have the correct sign *and* have magnitude at least 1 in order to be zero. Hinge loss has the useful property that it is an upper bound on error rate: for any sample S, the training error is at most $\mathrm{P}_{\mathbf{x}\in S} L(f_{\mathbf{w}}(\mathbf{x}),c^*(\mathbf{x}))$. With this loss function, stochastic gradient descent is called the *margin perceptron* algorithm.

More generally, we could have a much more complex class F. For example, consider a layered circuit of soft threshold gates. Each node in the circuit computes a linear function of its inputs and then passes this value through an "activation function" such as $a(z) = \tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$. This circuit could have multiple layers with the output of layer i being used as the input to layer $i + 1$. The vector $\mathbf{w}$ would be the concatenation of all the weight vectors in the network. This is the idea of *deep neural networks* discussed further in Section 5.15.

While it is difficult to give general guarantees on when stochastic gradient descent will succeed in finding a hypothesis of low error on its training set S, Theorems 5.7 and 5.5 imply that if it does and if S is sufficiently large, we can be confident that its true error will be low as well. Suppose that stochastic gradient descent is run on a machine where each weight is a 64-bit floating point number. This means that its hypotheses can each be described using $64n$ bits. If S has size at least $\frac{1}{\epsilon}[64n\ln(2) + \ln(1/\delta)]$, by Theorem 5.7 it is unlikely any such hypothesis of true error greater than will be consistent with the sample, and so if it finds a hypothesis consistent with S, we can be confident its true error is at most . Or, by Theorem 5.5, if $|S| \geq \frac{1}{2\epsilon^2}\left(64n\ln(2) + \ln(2/\delta)\right)$ then almost surely the final hypothesis h produced by stochastic gradient descent satisfies true error leas than or equal to training error plus .

5.14 Combining (Sleeping) Expert Advice

Imagine you have access to a large collection of rules-of-thumb that specify what to predict in different situations. For example, in classifying news articles, you might have one that says "if the article has the word 'football', then classify it as sports" and another that says "if the article contains a dollar figure, then classify it as business". In predicting the stock market, these could be different economic indicators. These predictors might at

times contradict each other, e.g., a news article that has both the word "football" and a dollar figure, or a day in which two economic indicators are pointing in different directions. It also may be that no predictor is perfectly accurate with some much better than others. We present here an algorithm for combining a large number of such predictors with the guarantee that if any of them are good, the algorithm will perform nearly as well as each good predictor on the examples on which that predictor fires.

Formally, define a "sleeping expert" to be a predictor h that on any given example **x** either makes a prediction on its label or chooses to stay silent (asleep). We will think of them as black boxes. Now, suppose we have access to n such sleeping experts $h_1,\ldots,h_n$, and let S_i denote the subset of examples on which h_i makes a prediction (e.g., this could be articles with the word "football" in them). We consider the online learning model, and let *mistakes*(A,S) denote the number of mistakes of an algorithm A on a sequence of examples S. Then the guarantee of our algorithm A will be that for all i

$$E\big(mistakes(A,S_i)\big) \leq (1+\epsilon)\cdot mistakes(h_i,S_i) + O\left(\tfrac{\log n}{\epsilon}\right)$$

where is a parameter of the algorithm and the expectation is over internal randomness in the randomized algorithm A.

As a special case, if $h_1,\ldots,h_n$ are concepts from a concept class H, and so they all make predictions on every example, then A performs nearly as well as the best concept in H. This can be viewed as a noise-tolerant version of the Halving Algorithm of Section 5.8.2 for the case that no concept in H is perfect. The case of predictors that make predictions on every example is called the problem of *combining expert advice*, and the more general case of predictors that sometimes fire and sometimes are silent is called the *sleeping experts* problem.

Combining Sleeping Experts Algorithm:

Initialize each expert h_i with a weight $w_i = 1$. Let $\epsilon \in (0,1)$. For each example x, do the following:

1. [Make prediction] Let H_x denote the set of experts h_i that make a prediction on x, and let $w_x = \sum_{h_j \in H_x} w_j$. Choose $h_i \in H_x$ with probability $p_{ix} = w_i/w_x$ and predict $h_i(x)$.

2. [Receive feedback] Given the correct label, for each $h_i \in H_x$ let $m_{ix} = 1$ if $h_i(x)$ was incorrect, else let $m_{ix} = 0$.

3. [Update weights] For each $h_i \in H_x$, update its weight as follows:

- Let $r_{ix} = \left(\sum_{h_j \in H_x} p_{jx} m_{jx}\right)/(1+\epsilon) - m_{ix}$.
- Update $w_i \leftarrow w_i(1+\epsilon)^{r_{ix}}$.

  Note that $\sum_{h_j \in H_x} p_{jx} m_{jx}$ represents the algorithm's probability of making a mistake on example x. So, h_i is rewarded for predicting correctly ($m_{ix} = 0$) especially when the algorithm had a high probability of making a mistake, and h_i is penalized for predicting incorrectly ($m_{ix} = 1$) especially when the algorithm had a low probability of making a mistake.

For each $h_i \not\in H_x$, leave w_i alone.

Theorem 5.22 *For any set of n sleeping experts $h_1,\ldots,h_n$, and for any sequence of examples S, the Combining Sleeping Experts Algorithm A satisfies for all i:*

$$E\big(mistakes(A,S_i)\big) \le (1+\epsilon)\cdot mistakes(h_i,S_i) + O\left(\tfrac{\log n}{\epsilon}\right)$$

where $S_i = \{x \in S : h_i \in H_x\}$.

Proof: Consider sleeping expert h_i. The weight of h_i after the sequence of examples S is exactly:

$$\begin{aligned} w_i &= (1+\epsilon)^{\sum_{x\in S_i}\left[\left(\sum_{h_j\in H_x} p_{jx}m_{jx}\right)/(1+\epsilon)-m_{ix}\right]} \\ &= (1+\epsilon)^{E[mistakes(A,S_i)]/(1+\epsilon)-mistakes(h_i,S_i)}. \end{aligned}$$

Let $w = \sum_j w_j$. Clearly $w_i \le w$. Therefore, taking logs, we have:

$$E\big(mistakes(A,S_i)\big)/(1+\epsilon) - mistakes(h_i,S_i) \;\le\; \log_{1+\epsilon} w.$$

So, using the fact that $\log_{1+\epsilon} w = O(\frac{\log W}{\epsilon})$,

$$E\big(mistakes(A,S_i)\big) \;\le\; (1+\epsilon)\cdot mistakes(h_i,S_i) + O\left(\tfrac{\log w}{\epsilon}\right).$$

Initially, $w = n$. To prove the theorem, it is enough to prove that w never increases. To do so, we need to show that for each x, $\sum_{h_i\in H_x} w_i(1+\epsilon)^{r_{ix}} \le \sum_{h_i\in H_x} w_i$, or equivalently dividing both sides by $\sum_{h_j\in H_x} w_j$ that $\sum_i p_{ix}(1+\epsilon)^{r_{ix}} \le 1$, where for convenience we define $p_{ix} = 0$ for $h_i \not\in H_x$.

For this we will use the inequalities that for $\beta, z \in [0,1]$, $\beta^z \le 1-(1-\beta)z$ and $\beta^{-z} \le 1+(1-\beta)z/\beta$. Specifically, we will use $\beta = (1+\epsilon)^{-1}$. We now have:

$$\begin{aligned}
\sum_i p_{ix}(1+\epsilon)^{r_{ix}} &= \sum_i p_{ix}\beta^{m_{ix}-(\sum_j p_{jx}m_{jx})\beta} \\
&\leq \sum_i p_{ix}\Big(1-(1-\beta)m_{ix}\Big)\left(1+(1-\beta)\left(\sum_j p_{jx}m_{jx}\right)\right) \\
&\leq \left(\sum_i p_{ix}\right) - (1-\beta)\sum_i p_{ix}m_{ix} + (1-\beta)\sum_i p_{ix}\sum_j p_{jx}m_{jx} \\
&= 1-(1-\beta)\sum_i p_{ix}m_{ix} + (1-\beta)\sum_j p_{jx}m_{jx} \\
&= 1,
\end{aligned}$$

where the second-to-last line follows from using $\sum_i p_{ix} = 1$ in two places. So w never increases and the bound follows as desired. ■

5.15 Deep Learning

Deep learning, or *deep neural networks*, refers to training many-layered networks of nonlinear computational units.

Each computational unit or gate works as follows: there are a set of "wires" bringing inputs to the gate. Each wire has a "weight"; the gate's output is a real number obtained by applying a non-linear "activation function" $g : \mathbf{R} \to \mathbf{R}$ to the the weighted sum of the input values. The activation function g is generally the same for all gates in the network, though, the number of inputs to individual gates may differ.

The input to the network is an example $\mathbf{x} \in R^d$. The first layer of the network transforms the example into a new vector $f_1(\mathbf{x})$. Then the second layer transforms $f_1(\mathbf{x})$ into a new vector $f_2(f_1(\mathbf{x}))$, and so on. Finally, the k^{th} layer outputs the final prediction $f(\mathbf{x}) = f_k(f_{k-1}(\ldots(f_1(\mathbf{x}))))$.

In supervised learning, we are given training examples $\mathbf{x_1}, \mathbf{x_2}, \ldots$, and corresponding labels $c^*(\mathbf{x_1}), c^*(\mathbf{x_2}), \ldots$. The training process finds a set of weights of all wires so as to minimize the error: $(f_0(\mathbf{x_1}) - c^*(\mathbf{x_1}))^2 + (f_0(\mathbf{x_2}) - c^*(\mathbf{x_2}))^2 + \cdots$. (One could alternatively aim to minimize other quantities besides the sum of squared errors of training examples.) Often training is carried out by running stochastic gradient descient, i.e., doing stochastic gradient descent in the weights space.

The motivation for deep learning is that often we are interested in data, such as images, that are given to us in terms of very low-level features, such as pixel intensity values. Our goal is to achieve some higher-level understanding of each image, such as what objects are in the image and what they are doing. To do so, it is natural to first convert the given low-level representation into one of higher-level features. That is what the layers of the network aim to do. Deep learning is also motivated by multi-task learning, with the idea

that a good higher-level representation of data should be useful for a wide range of tasks. Indeed, a common use of deep learning for multi-task learning is to share initial levels of the network across tasks.

A typical architecture of a deep neural network consists of layers of logic units. In a fully connected layer, the output of each gate in the layer is connected to the input of every gate in the next layer. However, if the input is an image one might like to recognize features independent of where they are located in the image. To achieve this one often uses a number of convolution layers. In a convolution layer, each gate gets inputs from a small $k \times k$ grid where k may be 5 to 10. There is a gate for each $k \times k$ square array of the image. The weights on each gate are tied together so that each gate recognizes the same feature. There will be several such collections of gates, so several different features can be learned. Such a level is called a convolution level and the fully connected layers are called autoencoder levels. A technique called *pooling* is used to keep the number of gates reasonable. A small $k \times k$ grid with k typically set to two is used to scan a layer. The stride is set so the grid will provide a non overlapping cover of the layer. Each $k \times k$ input grid will be reduced to a single cell by selecting the maximum input value or the average of the inputs. For $k = 2$ this reduces the number of cells by a factor of four.

Deep learning networks are trained by stochastic gradient descent (Section 5.13), sometimes called back propagation in the network context. An error function is constructed and the weights are adjusted using the derivative of the error function. This requires that the error function be differentiable. A smooth threshold is used such as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad \text{where} \quad \frac{\partial}{\partial x}\frac{e^e - e^{-e}}{e^x + e^{-x}} = 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2$$

or $\text{sigmod}(x) = \frac{1}{1+e^{-x}}$ where

$$\frac{\partial\, \text{sigmod}(x)}{\partial x} = \frac{e^{-x}}{(1 + e^{-x})^2} = sigmod(x)\frac{e^{-x}}{1 + e^{-x}} = \text{sigmoid}(x)\,(1 - \text{sigmoid}(x))$$

In fact the function

$$ReLU(x) = \begin{cases} x & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{where} \quad \frac{\partial \text{ReLU}(x)}{\partial x} = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

seems to work well even though its derivative at $x = 0$ is undefined. An advantage of ReLU over sigmoid is that ReLU does not saturate far from the origin.

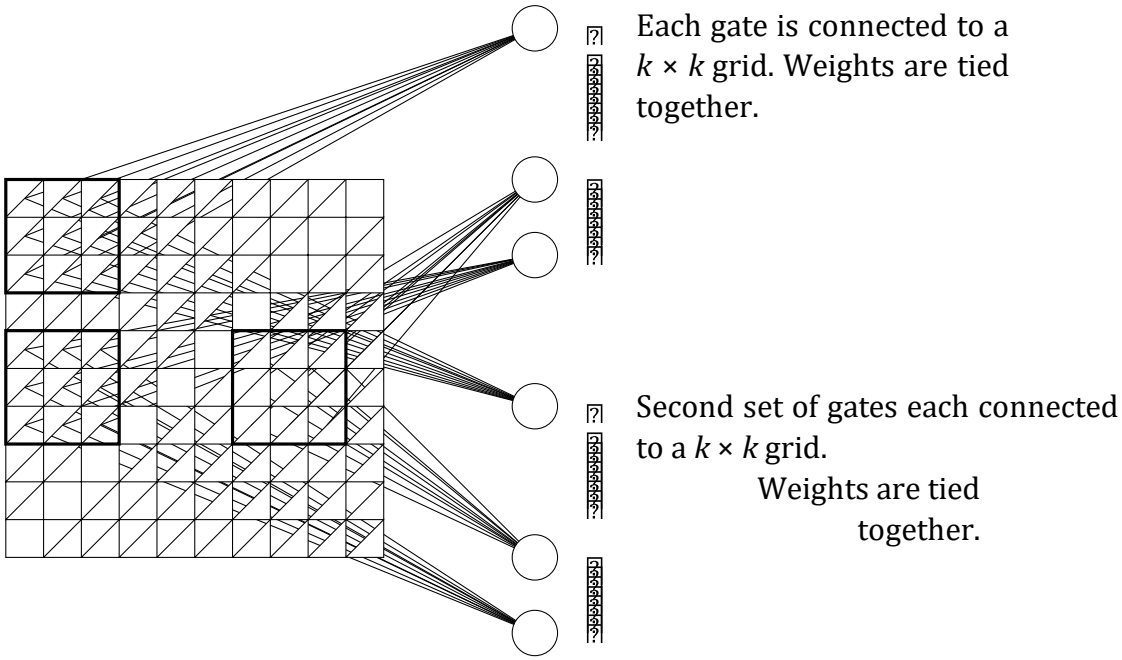


Figure 5.7: Convolution layers

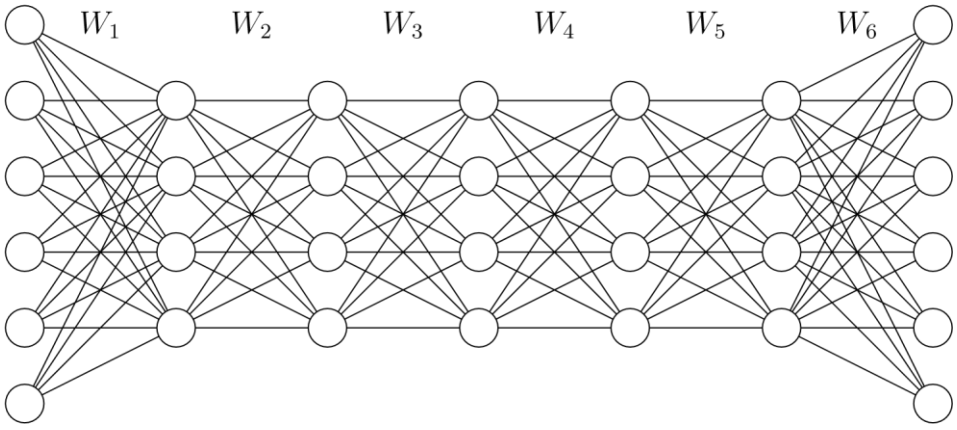


Figure 5.8: A deep learning fully connected network.

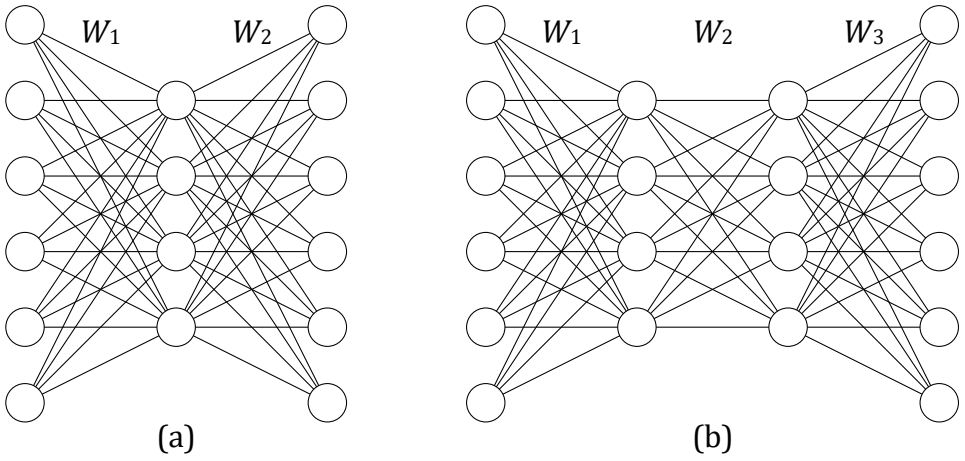


Figure 5.9: Autoencoder technique used to train one level at a time. In the Figure 5.9 (a) train W_1 and W_2. Then in Figure 5.9 (b), freeze W_1 and train W_2 and W_3. In this way one trains one set of weights at a time.

Training a deep learning network of 7 or 8 levels using gradient descent can be computationally expensive.[26] To address this issue one can train one level at a time on unlabeled data using an idea called autoencoding. There are three levels, the input, a middle level called the hidden level, and an output level as shown in Figure 5.9a. There are two sets of weights. W_1 is the weights of the hidden level gates and W_2 is W_1^T. Let **x** be the input pattern and **y** be the output. The error is $|\mathbf{x} - \mathbf{y}|^2$. One uses gradient descent to reduce the error. Once the weights W_1 are determined they are frozen and a second hidden level of gates is added as in Figure 5.9 b. In this network $W_3 = W_2^T$ and stochastic gradient descent is again used this time to determine W_2. In this way one level of weights is trained at a time.

The output of the hidden gates is an encoding of the input. An image might be a 10^8 dimensional input and there may only be 10^5 hidden gates. However, the number of images might be 10^7 so even though the dimension of the hidden layer is smaller than the dimension of the input, the number of possible codes far exceeds the number of inputs and thus the hidden layer is a compressed representation of the input. If the hidden layer were the same dimension as the input layer one might get the identity mapping. This does not happen for gradient descent starting with random weights.

The output layer of a deep network typically uses a softmax procedure. Softmax is a generalization of logistic regression where given a set of vectors $\{\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_n}\}$ with labels

[26] In the image recognition community, researchers work with networks of 150 levels. The levels tend to be convolution rather than fully connected.

$l_1, l_2, \ldots l_n$, $l_i \in \{0,1\}$ and with a weight vector $\mathbf{w}$ we define the probability that the label l given x equals 0 or 1 by

$$\text{Prob}(l = 1|\mathbf{x}) = \frac{1}{e^{-\mathbf{w}^T\mathbf{x}}} = \sigma(\mathbf{w}^T\mathbf{x})$$

and

$$\text{Prob}(l = 0|\mathbf{x}) = 1 - \text{Prob}(l = 1/\mathbf{x})$$

where σ is the sigmoid function.

Define a cost function

$$J(\mathbf{w}) = \sum_i \left(l_i \log(\text{Prob}(l = 1|\mathbf{x})) + (1 - l_i)\log(1 - \text{Prob}(l = 1|\mathbf{x})) \right)$$

and compute $\mathbf{w}$ to minimize $J(\mathbf{x})$. Then

$$J(\mathbf{w}) = \sum_i \left(l_i \log(\sigma(\mathbf{w}^T\mathbf{x})) + (1 - l_i)\log(1 - \sigma(\mathbf{w}^T\mathbf{x})) \right)$$

Since $\frac{\partial \sigma(\mathbf{w}^T\mathbf{x})}{\partial w_j} = \sigma(\mathbf{w}^T\mathbf{x})(1 - \sigma(\mathbf{w}^T\mathbf{x}))x_j$, it follows that $\frac{\partial \log(\sigma(\mathbf{w}^T\mathbf{x}))}{\partial w_j} = \frac{\sigma(\mathbf{w}^T\mathbf{x})(1-\sigma(\mathbf{w}^T\mathbf{x}))x_j}{\sigma(\mathbf{w}^T\mathbf{x})}$,
Thus

$$\frac{\partial J}{\partial w_j} = \sum_i \left(l_i \frac{\sigma(\mathbf{w}^T\mathbf{x})(1 - \sigma(\mathbf{w}^T\mathbf{x}))\mathbf{w}^T\mathbf{x})}{\sigma(\mathbf{w}^T\mathbf{x})} x_j - (1 - l_i)\frac{(1 - \sigma(\mathbf{w}^T\mathbf{x}))\sigma(\mathbf{w}^T\mathbf{x})}{1 - \sigma(\mathbf{w}^T\mathbf{x})} x_j \right)$$

$$= \sum_i \left(l_i(1 - \sigma(\mathbf{w}^T\mathbf{x}))x_j - (1 - l_i)\sigma(\mathbf{w}^T\mathbf{x}) x_j \right)$$

$$= \sum_i \left((l_i x_j - l_i\sigma(\mathbf{w}^T\mathbf{x})x_j - \sigma(\mathbf{w}^T\mathbf{x})x_j + l_i\sigma(\mathbf{w}^T\mathbf{x}) x_j \right)$$

$$= \sum_i \left(l_i - \sigma(\mathbf{w}^T\mathbf{x}) \right) x_j.$$

Softmax is a generalization of logistic regression to multiple classes. Thus, the labels l_i take on values $\{1,2,\ldots,k\}$. For an input $\mathbf{x}$, softmax estimates the probability of each label. The hypothesis is of the form

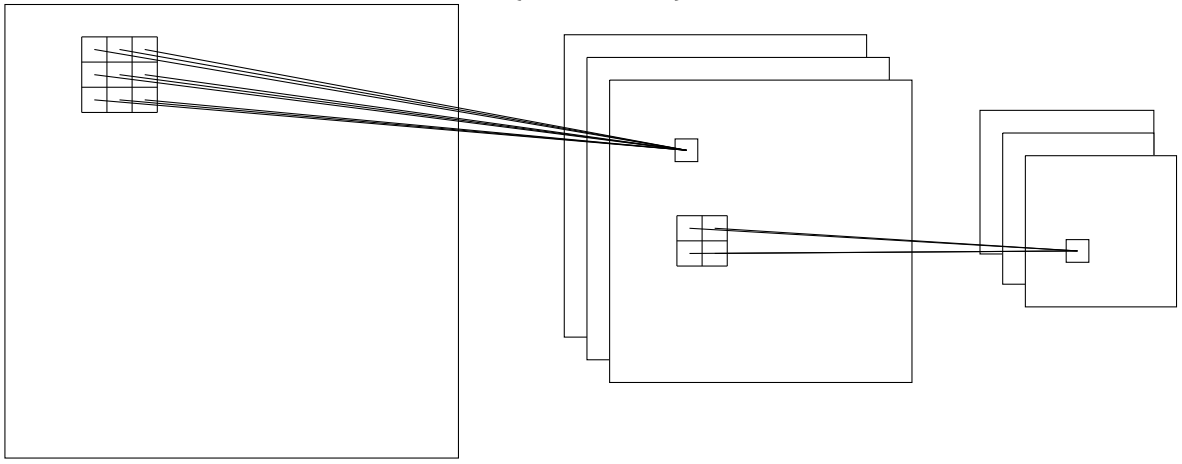
$$\begin{bmatrix} \text{Prob}(l = 1|\mathbf{x},\mathbf{w_1}) \\ \text{Prob}(l = 2|\mathbf{x},\mathbf{w_2}) \end{bmatrix} \quad \frac{1}{\begin{bmatrix} e^{\mathbf{w_1}^T\mathbf{x}} \\ e^{\mathbf{w_2}^T\mathbf{x}} \end{bmatrix}}$$

$h_w(x)$ = □□□ ... □ ——————— | = $\sum_{i=1}^{k} e$ wiTx □□□ ... □□□

Prob($l = k|\mathbf{x}, \mathbf{w_k}$) $e_{\mathbf{w_k T x}}$

where the matrix formed by the weight vectors is

$W = (\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_k})^T$



convolution pooling

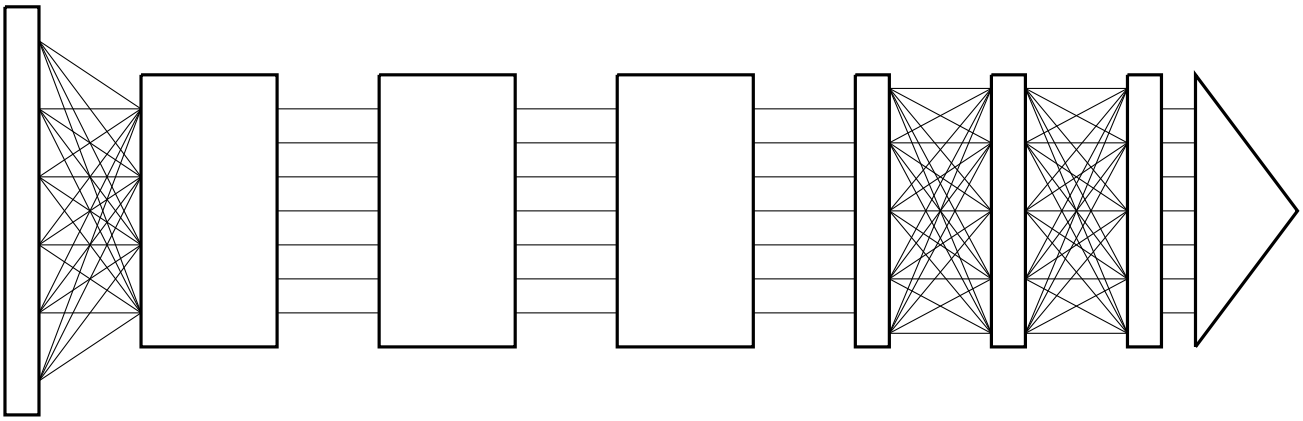


Image Convolutionlevels Fullyconnectedlevels Softmax

Figure 5.10: A convolution network W is a matrix since for each label l_i, there is a vector $\mathbf{w_i}$ of weights.

Consider a set of n inputs $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$. Define

$$\delta(l = k) = \begin{cases} 1 & \text{if } l = k \\ 0 & \text{otherwise} \end{cases}$$

and

$$J(W) = \sum_{i=1}^{n} \sum_{j=1}^{k} \delta(l_i = j) \log \frac{e^{\mathbf{w_j}^T x_i}}{\sum_{h=1}^{k} e^{\mathbf{w_h}^T x_i}}.$$

The derivative of the cost function with respect to the weights is

$$\nabla_{\mathbf{w_i}} J(W) = -\sum_{j=1}^{n} \mathbf{x_j} \left(\delta(l_j = k) - \text{Prob}(l_j = k) | \mathbf{x_j}, W \right).$$

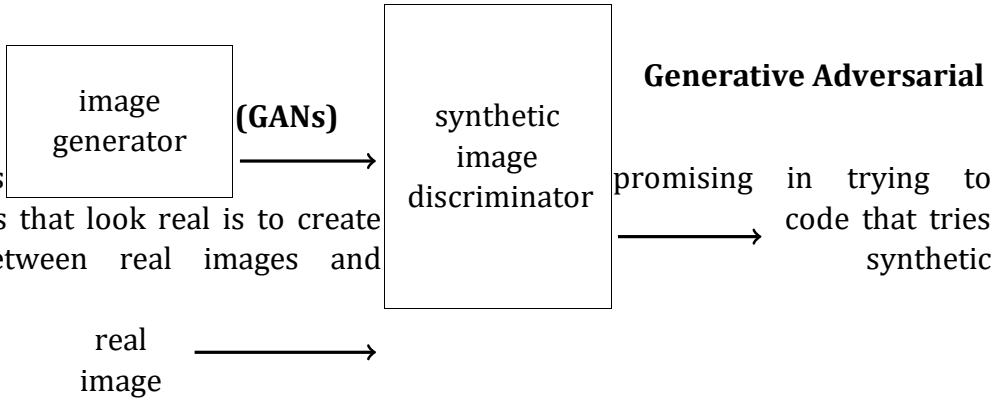
Note $\nabla_{\mathbf{w_i}} J(W)$ is a vector. Since $\mathbf{w_i}$ is a vector, each component of $\nabla_{\mathbf{w_i}} J(W)$ is the derivative with respect to one component of the vector $\mathbf{w_i}$.

Over fitting is a major concern in deep learning since large networks can have hundreds of millions of weights. In image recognition, the number of training images can be significantly increased by random jittering of the images. Another technique called *dropout* randomly deletes a fraction of the weights at each training iteration. Regularization is used to assign a cost to the size of weights and many other ideas are being explored.

Deep learning is an active research area. Some of the ideas being explored are what do individual gates or sets of gates learn. If one trains a network twice from starting with random sets of weights, do gates learn the same features? In image recognition, the early convolution layers seem to learn features of images rather than features of the specific set of images they are being trained with. Once a network is trained on say a set of images one of which is a cat one can freeze the weights and then find images that will map to the activation vector generated by the cat image. One can take an artwork image and separate the style from the content and then create an image using the content but a different style [GEB15]. This is done by taking the activation of the original image and moving it to the manifold of activation vectors of images of a given style. One can do many things of this type. For example one can change the age of a child in an image or change some other feature [GKL+15]. For more information about deep learning, see

[Ben09].[27]

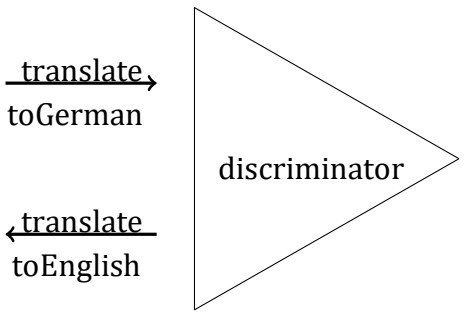
5.15.1 Generative Adversarial Networks (GANs)



A method that is promising in trying to generate images that look real is to create code that tries to discern between real images and synthetic images.

One first trains the synthetic image discriminator to distinguish between real images and synthetic ones. Then one trains the image generator to generate images that the discriminator believes are real images. Alternating the training between the two units ends up forcing the image generator to produce real looking images. This is the idea of Generative Adversarial Networks.

There are many possible applications for this technique. Suppose you wanted to train a network to translate from English to German. First train a discriminator to determine if a sentence is a real sentence in German as opposed to a synthetic sentence. Then train a translator for English to German and a translator from German to English.



5.16 Further Current Directions

We now briefly discuss a few additional current directions in machine learning, focusing on *semi-supervised* learning, *active* learning, and *multi-task* learning.

[27] See also the tutorials: http://deeplearning.net/tutorial/deeplearning.pdf and http://deeplearning.stanford.edu/tutorial/.

5.16.1 Semi-Supervised Learning

Semi-supervised learning refers to the idea of trying to use a large unlabeled data set U to augment a given labeled data set L in order to produce more accurate rules than would have been achieved using just L alone. The motivation is that in many settings (e.g., document classification, image classification, speech recognition), unlabeled data is much more plentiful than labeled data, so one would like to make use of it if possible. Of course, unlabeled data is missing the labels! Nonetheless it often contains information that an algorithm can take advantage of.

As an example, suppose one believes the target function is a linear separator that separates most of the data by a large margin. By observing enough unlabeled data to estimate the probability mass near to any given linear separator, one could in principle then discard separators in advance that slice through dense regions and instead focus attention on just those that indeed separate most of the distribution by a large margin. This is the high level idea behind a technique known as Semi-Supervised SVMs. Alternatively, suppose data objects can be described by two different "kinds" of features (e.g., a webpage could be described using words on the page itself or using words on links pointing *to* the page), and one believes that each kind should be sufficient to produce an accurate classifier. Then one might want to train a *pair* of classifiers (one on each type of feature) and use unlabeled data for which one is confident but the other is not to bootstrap, labeling such examples with the confident classifier and then feeding them as training data to the less-confident one. This is the high-level idea behind a technique known as Co-Training. Or, if one believes "similar examples should generally have the same label", one might construct a graph with an edge between examples that are sufficiently similar, and aim for a classifier that is correct on the labeled data and has a small cut value on the unlabeled data; this is the high-level idea behind graph-based methods.

A formal model: The batch learning model introduced in Sections 5.1 and 5.6 in essence assumes that one's prior beliefs about the target function be described in terms of a class of functions H. In order to capture the reasoning used in semi-supervised learning, we need to also describe beliefs about the *relation* between the target function and the data distribution. A clean way to do this is via a *notion of compatibility* χ between a hypothesis h and a distribution D. Formally, χ maps pairs (h,D) to $[0,1]$ with $\chi(h,\mathrm{D}) = 1$ meaning that h is highly compatible with D and $\chi(h,\mathrm{D}) = 0$ meaning that h is very *in*compatible with D. The quantity $1-\chi(h,\mathrm{D})$ is called the *unlabeled error rate* of h, and denoted $err_{unl}(h)$. Note that for χ to be useful, it must be estimatable from a finite sample; to this end, let us further require that χ is an expectation over individual examples. That is, overloading notation for convenience, we require $\chi(h,\mathrm{D}) = \mathbf{E}_{x\sim \mathrm{D}}[\chi(h,x)]$, where $\chi : \mathrm{H} \times \mathrm{X} \to [0,1]$.

For instance, suppose we believe the target should separate most data by margin γ. We can represent this belief by defining $\chi(h,x) = 0$ if x is within distance γ of the decision boundary of h, and $\chi(h,x) = 1$ otherwise. In this case, $err_{unl}(h)$ will denote the probability mass of D within distance γ of h's decision boundary. As a different example, in co-training, we assume each example can be described using two "views" that each are sufficient for classification; that is, there exist c_1^*, c_2^* such that for each example $x = \langle x_1, x_2 \rangle$ we have $c_1^*(x_1) = c_2^*(x_2)$. We can represent this belief by defining a hypothesis $h = \langle h_1, h_2 \rangle$ to be compatible with an example $\langle x_1, x_2 \rangle$ if $h_1(x_1) = h_2(x_2)$ and incompatible otherwise; $err_{unl}(h)$ is then the probability mass of examples on which h_1 and h_2 disagree.

As with the class H, one can either assume that the target is fully compatible (i.e., $err_{unl}(c^*) = 0$) or instead aim to do well as a function of how compatible the target is. The case that we assume $c^* \in H$ and $err_{unl}(c^*) = 0$ is termed the "doubly realizable case". The concept class H and compatibility notion χ are both viewed as *known*.

Intuition: In this framework, the way that unlabeled data helps in learning can be intuitively described as follows. Suppose one is given a concept class H (such as linear separators) and a compatibility notion χ (such as penalizing h for points within distance γ of the decision boundary). Suppose also that one believes $c^* \in H$ (or at least is close) and that $err_{unl}(c^*) = 0$ (or at least is small). Then, unlabeled data can help by allowing one to estimate the *unlabeled error rate* of all $h \in H$, thereby in principle reducing the search space from H (all linear separators) down to just the subset of H that is highly compatible with D. The key challenge is how this can be done efficiently (in theory, in practice, or both) for natural notions of compatibility, as well as identifying types of compatibility that data in important problems can be expected to satisfy.

A theorem: The following is a semi-supervised analog of our basic sample complexity theorem, Theorem 5.3. First, fix some set of functions H and compatibility notion χ. Given a labeled sample L, define $err(h)$ to be the fraction of mistakes of h on L. Given c an unlabeled sample U, define $\chi(h,U) = \mathbf{E}_{x \sim U}[\chi(h,x)]$ and define $err_{unl}(h) = 1 - \chi(h,U)$. c That is, $err(h)$ and $err_{unl}(h)$ are the empirical error rate and unlabeled error rate of h, c c respectively. Finally, given $\alpha > 0$, define $H_{D,\chi}(\alpha)$ to be the set of functions $f \in H$ such that $err_{unl}(f) \leq \alpha$.

Theorem 5.23 *If $c^* \in H$ then with probability at least $1 - \delta$, for labeled set L and unlabeled set U drawn from D, the $h \in H$ that optimizes $err_{unl}(h)$ subject to $err(h) = 0$* c c *will have $err_{\mathcal{D}}(h) \leq \epsilon$ for*

$$|U| \geq \frac{2}{\epsilon^2}\left[\ln|\mathcal{H}| + \ln\frac{4}{\delta}\right], \text{ and } |L| \geq \frac{1}{\epsilon}\left[\ln|\mathcal{H}_{\mathcal{D},\chi}(err_{unl}(c^*) + 2\epsilon)| + \ln\frac{2}{\delta}\right].$$

Equivalently, for |U| satisfying this bound, for any |L|, whp the h ∈ H *that minimizes* $err_{unl}(h)$ *subject to* $err(h) = 0$ *has* c c

$$err_{\mathcal{D}}(h) \leq \frac{1}{|L|}\left[\ln|\mathcal{H}_{\mathcal{D},\chi}(err_{unl}(c^*)+2\epsilon)| + \ln\frac{2}{\delta}\right].$$

Proof: By Hoeffding bounds, |*U*| is sufficiently large so that with probability at least $1 - \delta/2$, all $h \in H$ have $|\widehat{err}_{unl}(h) - err_{unl}(h)| \leq \epsilon$. Thus we have:

$$\{f \in \mathcal{H} : \widehat{err}_{unl}(f) \leq err_{unl}(c^*) + \epsilon\} \subseteq \mathcal{H}_{\mathcal{D},\chi}(err_{unl}(c^*) + 2\epsilon).$$

The given bound on |*L*| is sufficient so that with probability at least $1-\delta$, all $h \in H$ with $err(h) = 0$ and $\widehat{err}_{unl}(h) \leq err_{unl}(c^*) + \epsilon$ have $err_{\mathcal{D}}(h) \leq \epsilon$; furthermore, $err_{unl}(c^*) \leq$ cc $err_{unl}(c^*) + \epsilon$, so such a function *h* exists. Therefore, with probability at least $1 - \delta$, the $h \in H$ that optimizes $err_{unl}(h)$ subject to $err(h) = 0$ has $err_{\mathcal{D}}(h) \leq \epsilon$, as desired. ■c c
One can view Theorem 5.23 as bounding the number of labeled examples needed to learn well as a function of the "helpfulness" of the distribution D with respect to χ. Namely, a helpful distribution is one in which $H_{D,\chi}(\alpha)$ is small for α slightly larger than the compatibility of the true target function, so we do not need much labeled data to identify a good function among those in $H_{D,\chi}(\alpha)$. For more information on semi-supervised learning, see [BB10, BM98, CSZ06, Joa99, Zhu06, ZGL03].

5.16.2 Active Learning

Active learning refers to algorithms that take an active role in the selection of which examples are labeled. The algorithm is given an initial unlabeled set *U* of data points drawn from distribution D and then interactively requests for the labels of a small number of these examples. The aim is to reach a desired error rate using much fewer labels than would be needed by just labeling random examples (i.e., passive learning).

As a simple example, suppose that data consists of points on the real line and $H = \{f_a : f_a(x) = 1 \text{ iff } x \geq a\}$ for $a \in R$. That is, H is the set of all threshold functions on the line. It is not hard to show (see Exercise 5.2) that a random labeled sample of size $O(\frac{1}{\epsilon}\log(\frac{1}{\delta}))$ is sufficient to ensure that with probability $\geq 1 - \delta$, any consistent threshold a^0 has error at most . Moreover, it is not hard to show that $\Omega(\frac{1}{\epsilon})$ random examples are necessary for passive learning. However, with active learning we can achieve error using only $O(\log(\frac{1}{\epsilon}) + \log\log(\frac{1}{\delta}))$ labels. Specifically, first draw an unlabeled sample *U* of size $O(\frac{1}{\epsilon}\log(\frac{1}{\delta}))$. Then query the leftmost and rightmost points: if these are both negative then output $a^0 = \infty$, and if these are both positive then output $a^0 = -\infty$. Otherwise (the leftmost is negative and the rightmost is positive), perform binary search to find two adjacent examples x, x^0 such that *x* is negative and x^0 is positive, and output $a^0 = (x+x^0)/2$. This threshold a^0 is consistent with the labels on the entire set *U*, and so by the above argument, has error ≤ with probability $\geq 1 - \delta$.

The agnostic case, where the target need not belong in the given class H is quite a bit more subtle, and addressed in a quite general way in the "A^2" Agnostic Active learning algorithm [BBL09]. For more information on active learning, see [Das11, BU14].

5.16.3 Multi-Task Learning

In this chapter we have focused on scenarios where our goal is to learn a single target function c^*. However, there are also scenarios where one would like to learn *multiple* target functions $c_1^*, c_2^*, \ldots, c_n^*$. If these functions are related in some way, then one could hope to do so with less data per function than one would need to learn each function separately. This is the idea of *multi-task learning*.

One natural example is object recognition. Given an image $\mathbf{x}$, $c_1^*(\mathbf{x})$ might be 1 if $\mathbf{x}$ is a coffee cup and 0 otherwise; $c_2^*(\mathbf{x})$ might be 1 if $\mathbf{x}$ is a pencil and 0 otherwise; $c_3^*(\mathbf{x})$ might be 1 if $\mathbf{x}$ is a laptop and 0 otherwise. These recognition tasks are related in that image features that are good for one task are likely to be helpful for the others as well. Thus, one approach to multi-task learning is to try to learn a common representation under which each of the target functions can be described as a simple function. Another natural example is personalization. Consider a speech recognition system with n different users. In this case there are n target tasks (recognizing the speech of each user) that are clearly related to each other. Some good references for multi-task learning are [TM95, Thr96].

5.17 Bibliographic Notes

The basic theory underlying learning in the distributional setting was developed by Vapnik [Vap82], Vapnik and Chervonenkis [VC71], and Valiant [Val84]. The connection of this to the notion of Occam's razor is due to [BEHW87]. For more information on uniform convergence, regularization and complexity penalization, see [Vap98]. The Perceptron algorithm for online learning of linear separators was first analyzed by Block [Blo62] and Novikoff [Nov62]; the proof given here is from [MP69]. A formal description of the online learning model and its connections to learning in the distributional setting is given in [Lit87]. Support Vector Machines and their connections to kernel functions were first introduced by [BGV92], and extended by [CV95], with analysis in terms of margins given by [STBWA98]. For further reading on SVMs, learning with kernel functions, and regularization, see [SS01]. VC dimension is due to Vapnik and Chervonenkis [VC71] with the results presented here given in Blumer, Ehrenfeucht, Haussler and Warmuth [BEHW89]. Boosting was first introduced by Schapire [Sch90], and Adaboost and its guarantees are due to Freund and Schapire [FS97] . Analysis of the problem of combining expert advice was given by Littlestone and Warmuth [LW94] and Cesa-Bianchi et al. [CBFH+97]; the analysis of the sleeping experts problem given here is from [BM07].

5.18 Exercises

Exercise 5.1 (Section 5.5 and 5.6) *Consider the instance space* $X = \{0,1\}^d$ *and let* H *be the class of 3-CNF formulas. That is,* H *is the set of concepts that can be described as a conjunction of clauses where each clause is an OR of up to 3 literals. (These are also called 3-SAT formulas). For example* c^* *might be* $(x_1 \vee \bar{x}_2 \vee x_3)(x_2 \vee x_4)(\bar{x}_1 \vee x_3)(x_2 \vee x_3 \vee x_4)$*. Assume we are in the PAC learning setting, so examples are drawn from some underlying distribution D and labeled by some 3-CNF formula* c^**.*

1. *Give a number of samples m that would be sufficient to ensure that with probability* $\geq 1 - \delta$*, all 3-CNF formulas consistent with the sample have error at most with respect to D.*

2. *Give a polynomial-time algorithm for PAC-learning the class of 3-CNF formulas.*

Exercise 5.2 (Section 5.5) *Consider the instance space* $X = R$*, and the class of functions* $H = \{f_a : f_a(x) = 1 \text{ iff } x \geq a\}$ *for* $a \in R$*. That is,* H *is the set of all threshold functions on the line. Prove that for any distribution* D*, a sample S of size* $O(\frac{1}{\epsilon}\log(\frac{1}{\delta}))$ *is sufficient to ensure that with probability* $\geq 1 - \delta$*, any* f_{a0} *such that* $err_S(f_{a0}) = 0$ *has* $err_{\mathcal{D}}(f_{a'}) \leq \epsilon$*. Note that you can answer this question from first principles, without using the concept of VC-dimension.*

Exercise 5.3 (Perceptron; Section 5.8.3) *Consider running the Perceptron algorithm in the online model on some sequence of examples S. Let* S^0 *be the same set of examples as S but presented in a different order. Does the Perceptron algorithm necessarily make the same number of mistakes on S as it does on* S^0*? If so, why? If not, show such an S and* S^0 *(consisting of the same set of examples in a different order) where the Perceptron algorithm makes a different number of mistakes on* S^0 *than it does on S.*

Exercise 5.4 (representation and linear separators) *Show that any disjunction (see Section 5.6.1) over* $\{0,1\}^d$ *can be represented as a linear separator. Show that moreover* $\sqrt{}$ *the margin of separation is* $\Omega(1/\sqrt{d})$*.*

Exercise 5.5 (Linear separators; easy) *Show that the parity function on* $d \geq 2$ *Boolean variables cannot be represented by a linear threshold function. The parity function is 1 if and only if an odd number of inputs is 1.*

Exercise 5.6 (Perceptron; Section 5.8.3) *We know the Perceptron algorithm makes at most* $1/\gamma^2$ *mistakes on any sequence of examples that is separable by margin* γ *(we assume all examples are normalized to have length 1). However, it need not find a separator of large margin. If we also want to find a separator of large margin, a natural alternative is to update on any example* $\mathbf{x}$ *such that* $f^*(\mathbf{x})(\mathbf{w} \cdot \mathbf{x}) < 1$*; this is called the* margin perceptron *algorithm.*

1. *Argue why margin perceptron is equivalent to running stochastic gradient descent on the class of linear predictors ($f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$) using hinge loss as the loss function and using $\lambda_t = 1$.*
2. *Prove that on any sequence of examples that are separable by margin γ, this algorithm will make at most $3/\gamma^2$ updates.*

3. *In part 2 you probably proved that each update increases $|\mathbf{w}|^2$ by at most 3. Use this (and your result from part 2) to conclude that if you have a dataset S that is separable by margin γ, and cycle through the data until the margin perceptron algorithm makes no more updates, that it will find a separator of margin at least $\gamma/3$.*

Exercise 5.7 (Decision trees, regularization; Section 5.6) *Pruning a decision tree: Let S be a labeled sample drawn iid from some distribution* D *over* $\{0,1\}^n$*, and suppose we have used S to create some decision tree T. However, the tree T is large, and we are concerned we might be overfitting. Give a polynomial-time algorithm for* pruning *T that finds the pruning h of T that optimizes the right-hand-side of Corollary 5.8, i.e., that for a given* $\delta > 0$ *minimizes:*

$$err_S(h) + \sqrt{\frac{\text{size}(h)\ln(4) + \ln(2/\delta)}{2|S|}}.$$

To discuss this, we need to define what we mean by a "pruning" of T and what we mean by the "size" of h. A pruning h of T is a tree in which some internal nodes of T have been turned into leaves, labeled "+" or "−" depending on whether the majority of examples in S that reach that node are positive or negative. Let $size(h) = L(h)\log(n)$ *where* $L(h)$ *is the number of leaves in h.*

Hint #1: it is sufficient, for each integer $L = 1,2,\ldots,L(T)$*, to find the pruning of T with L leaves of lowest empirical error on S, that is,* $h_L = \text{argmin}_{h:L(h)=L} err_S(h)$*. Then you can just plug them all into the displayed formula above and pick the best one. Hint #2: use dynamic programming.*

Exercise 5.8 (Decision trees, sleeping experts; Sections 5.6, 5.14) *"Pruning" a Decision Tree Online via Sleeping Experts: Suppose that, as in the above problem, we are given a decision tree T, but now we are faced with a sequence of examples that arrive online. One interesting way we can make predictions is as follows. For each node v of T (internal node or leaf) create two sleeping experts: one that predicts positive on any example that reaches v and one that predicts negative on any example that reaches v. So, the total number of sleeping experts is* $O(L(T))$*.*

1. *Say why any pruning h of T, and any assignment of* $\{+,-\}$ *labels to the leaves of h, corresponds to a subset of sleeping experts with the property that exactly one sleeping expert in the subset makes a prediction on any given example.*

2. *Prove that for any sequence S of examples, and any given number of leaves L, if we run the sleeping-experts algorithm using* $\epsilon = \sqrt{\frac{L\log(L(T))}{|S|}}$*, then the expected error rate of the algorithm on S (the total number of mistakes of the algorithm divided by |S|) will be at most* $err_S(h_L) + O(\sqrt{\frac{L\log(L(T))}{|S|}})$*, where* $h_L = \text{argmin}_{h:L(h)=L} err_S(h)$ *is the pruning of T with L leaves of lowest error on S.*

3. *In the above question, we assumed L was given. Explain how we can remove this assumption and achieve a bound of* $\min_L \left[err_S(h_L) + O(\sqrt{\frac{L\log(L(T))}{|S|}})\right]$ *by instantiating*
   L(*T*) *copies of the above algorithm (one for each value of L) and then combining* these *algorithms using the experts algorithm (in this case, none of them will be sleeping).*

Exercise 5.9 Kernels; (Section 5.3) *Prove Theorem 5.2.*

Exercise 5.10 *What is the VC-dimension of right corners with axis aligned edges that are oriented with one edge going to the right and the other edge going up?*

Exercise 5.11 (VC-dimension; Section 5.11) *What is the VC-dimension V of the class* H *of axis-parallel boxes in* R^d*? That is,* $H = \{h_{\mathbf{a},\mathbf{b}} : \mathbf{a},\mathbf{b} \in R^d\}$ *where* $h_{\mathbf{a},\mathbf{b}}(\mathbf{x}) = 1$ *if* $a_i \le x_i \le b_i$ *for all* $i = 1,\ldots,d$ *and* $h_{\mathbf{a},\mathbf{b}}(\mathbf{x}) = -1$ *otherwise.*

1. *Prove that the VC-dimension is at least your chosen V by giving a set of V points that is shattered by the class (and explaining why it is shattered).*

2. *Prove that the VC-dimension is at most your chosen V by proving that no set of* $V + 1$ *points can be shattered.*

Exercise 5.12 (VC-dimension, Perceptron, and Margins; Sections 5.8.3, 5.11) *Say that a set of points S is* shattered by linear separators of margin *γ if every labeling of the points in S is achievable by a linear separator of margin at least γ. Prove that no set of* $1/\gamma^2 + 1$ *points in the unit ball is shattered by linear separators of margin γ. Hint: think about the Perceptron algorithm and try a proof by contradiction.*

Exercise 5.13 *(Linear separators) Suppose the instance space* X *is* $\{0,1\}^d$ *and consider the target function* c^* *that labels an example* **x** *as positive if the least index i for which* $x_i = 1$ *is odd, else labels* **x** *as negative. In other words,* $c^*(\mathbf{x})$ *= "if* $x_1 = 1$ *then positive else if* $x_2 = 1$ *then negative else if* $x_3 = 1$ *then positive else ... else negative". Show that the rule can be represented by a linear threshold function.*

Exercise 5.14 *(Linear separators; harder) Prove that for the problem of Exercise 5.13, we cannot have a linear separator with margin at least* $1/f(d)$ *where* $f(d)$ *is bounded above by a polynomial function of d.*

Exercise 5.15 *VC-dimension Prove that the VC-dimension of circles in the plane is three.*

Exercise 5.16 *VC-dimension Show that the VC-dimension of arbitrary right triangles in the plane is seven.*

Exercise 5.17 *VC-dimension Prove that the VC-dimension of triangles in the plane is seven.*

Exercise 5.18 *VC-dimension Prove that the VC dimension of convex polygons in the plane is infinite.*

Exercise 5.19 *At present there are many interesting research directions in deep learning that are being explored. This exercise focuses on whether gates in networks learn the same thing independent of the architecture or how the network is trained. On the web there are several copies of Alexnet that have been trained starting from different random initial weights. Select two copies and form a matrix where the columns of the matrix correspond to gates in the first copy of Alexnet and the rows of the matrix correspond to gates of the same level in the second copy. The* ij^{th} *entry of the matrix is the covariance of the activation of the* j^{th} *gate in the first copy of Alexnet with the* i^{th} *gate in the second copy. The covariance is the expected value over all images in the data set.*

1. *Match the gates in the two copies of the network using a bipartite graph matching algorithm. What is the fraction of matches that have a high covariance?*

2. *It is possible that there is no good one to one matching of gates but that some small set of gates in the first copy of the network learn what some small set of gates in the second copy learn. Explore a clustering technique to match sets of gates and carry out an experiment to do this.*

Exercise 5.20

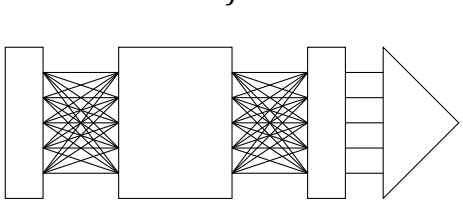
1. *Input an image to a deep learning network. Reproduce the image from the activation vector,* a_{image}*, it produced by inputting a random image and producing an activation vector* a_{random}*. Then by gradient descent modify the pixels in the random image to minimize the error function* $|a_{image} - a_{random}|^2$*.*

2. *Train a deep learning network to produce an image from an activation network.*

Exercise 5.21

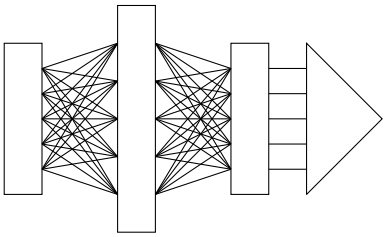
1. *Create and train a simple deep learning network consisting of a convolution level with pooling, a fully connected level, and then softmax. Keep the network small. For input data use the MNIST data set* http://yann.lecun.com/exdb/mnist/ with 28×28 *images of digits. Use maybe 20 channels for the convolution level and 100 gates for the fully connected level.*

2. *Create and train a second network with two fully connected levels, the first level with 200 gates and the second level with 100 gates. How does the accuracy of the second network compare to the first?*
3. *Train the second network again but this time use the activation vector of the 100 gate level and train the second network to produce that activation vector and only then train the softmax. How does the accuracy compare to direct training of the second network and the first network?*



convolution

first network



second network

6 Algorithms for Massive Data Problems: Streaming, Sketching, and Sampling

6.1 Introduction

This chapter deals with massive data problems where the input data is too large to be stored in random access memory. One model for such problems is the streaming model, where n data items $a_1,a_2,\ldots,a_n$ arrive one at a time. For example, the a_i might be IP addresses being observed by a router on the internet. The goal is for our algorithm to compute some statistics, property, or summary of these data items without using too much memory, much less than n. More specifically, we assume each a_i itself is a b-bit quantity where b is not too large. For example, each a_i might be an integer in $\{1,\ldots,m\}$ where $m = 2^b$. Our goal will be to produce some desired output using space polynomial in b and $\log n$; see Figure 6.1.

For example, a very easy problem to solve in the streaming model is to compute the sum of all the a_i. If each a_i is an integer between 1 and $m = 2^b$, then the sum of all the a_i is an integer between 1 and mn and so the number of bits of memory needed to maintain the sum is $O(b + \log n)$. A harder problem, which we discuss shortly, is computing the number of distinct numbers in the input sequence.

One natural approach for tackling a range of problems in the streaming model is to perform random sampling of the input "on the fly". To introduce the basic flavor of sampling on the fly, consider a stream $a_1,a_2,\ldots,a_n$ from which we are to select an index i with probability proportional to the value of a_i. When we see an element, we do not know the probability with which to select it since the normalizing constant depends on all of the elements including those we have not yet seen. However, the following method works. Let s be the sum of the a_i's seen so far. Maintain s and an index i selected with probability $\frac{a_i}{s}$. Initially $i = 1$ and $s = a_1$. Having seen symbols $a_1,a_2,\ldots,a_j$, s will equal $a_1 + a_2 + \cdots + a_j$ and for each i in $\{1,\ldots,j\}$, the selected index will be i with probability $\frac{a_i}{s}$. On seeing a_{j+1}, change the selected index to $j + 1$ with probability $\frac{a_{j+1}}{s+a_{j+1}}$ and otherwise keep the same index as before with probability $1 - \frac{a_{j+1}}{s+a_{j+1}}$. If we change the index to $j + 1$, clearly it was selected with the correct probability. If we keep i as our selection, then it will have been selected with probability

$$\left(1 - \frac{a_{j+1}}{s + a_{j+1}}\right)\frac{a_i}{s} = \frac{s}{s + a_{j+1}}\frac{a_i}{s} = \frac{a_i}{s + a_{j+1}}$$

which is the correct probability for selecting index i. Finally s is updated by adding a_{j+1} to s. This problem comes up in many areas such as sleeping experts where there is a sequence of weights and we want to pick an expert with probability proportional to its weight. The a_i's are the weights and the subscript i denotes the expert.

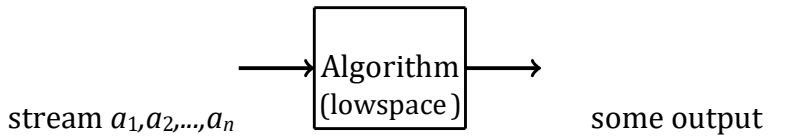


Figure 6.1: High-level representation of the streaming model

6.2 Frequency Moments of Data Streams

An important class of problems concerns the frequency moments of data streams. As mentioned above, a data stream $a_1,a_2,\ldots,a_n$ of length n consists of symbols a_i from an alphabet of m possible symbols which for convenience we denote as $\{1,2,\ldots,m\}$. Throughout this section, n,m, and a_i will have these meanings and s (for symbol) will denote a generic element of $\{1,2,\ldots,m\}$. The frequency f_s of the symbol s is the number of occurrences of s in the stream. For a nonnegative integer p, the p^{th} frequency moment of the stream is

$$\sum_{s=1}^{m} f_s^p.$$

Note that the $p = 0$ frequency moment corresponds to the number of distinct symbols occurring in the stream using the convention $0^0 = 0$. The first frequency moment is just n, the length of the string. The second frequency moment, $\sum_s f_s^2$, is useful in computing the variance of the stream, i.e., the average squared difference from the average frequency.

$$\frac{1}{m}\sum_{s=1}^{m}\left(f_s - \frac{n}{m}\right)^2 = \frac{1}{m}\sum_{s=1}^{m}\left(f_s^2 - 2\frac{n}{m}f_s + \left(\frac{n}{m}\right)^2\right) = \left(\frac{1}{m}\sum_{s=1}^{m} f_s^2\right) - \frac{n^2}{m^2}$$

In the limit as p becomes large, $\left(\sum_{s=1}^{m} f_s^p\right)^{1/p}$ is the frequency of the most frequent element(s).

We will describe sampling based algorithms to compute these quantities for streaming data shortly. First a note on the motivation for these problems. The identity and frequency of the the most frequent item, or more generally, items whose frequency exceeds a given fraction of n, is clearly important in many applications. If the items are packets on a network with source and/or destination addresses, the high frequency items identify the heavy bandwidth users. If the data consists of purchase records in a supermarket, the high frequency items are the best-selling items. Determining the number of distinct symbols is the abstract version of determining such things as the number of accounts, web users, or credit card holders. The second moment and variance are useful in networking as well as in database and other applications. Large amounts of network log data are generated by

routers that can record the source address, destination address, and the number of packets for all the messages passing through them. This massive data cannot be easily sorted or aggregated into totals for each source/destination. But it is important to know if some popular source-destination pairs have a lot of traffic for which the variance is one natural measure.

6.2.1 Number of Distinct Elements in a Data Stream

Consider a sequence $a_1,a_2,...,a_n$ of n elements, each a_i an integer in the range 1 to m where n and m are very large. Suppose we wish to determine the number of distinct a_i in the sequence. Each a_i might represent a credit card number extracted from a sequence of credit card transactions and we wish to determine how many distinct credit card accounts there are. Note that this is easy to do in $O(m)$ space by just storing a bit-vector that records which elements have been seen so far and which have not. It is also easy to do in $O(n\log m)$ space by storing a list of all distinct elements that have been seen. However, our goal is to use space logarithmic in m and n. We first show that this is impossible using an exact deterministic algorithm. Any deterministic algorithm that determines the number of distinct elements exactly must use at least m bits of memory on some input sequence of length $O(m)$. We then will show how to get around this problem using randomization and approximation.

Lower bound on memory for exact deterministic algorithm

We show that any exact deterministic algorithm must use at least m bits of memory on some sequence of length $m+1$. Suppose we have seen $a_1,...,a_m$, and suppose for sake of contradiction that our algorithm uses less than m bits of memory on all such sequences. There are $2^m - 1$ possible subsets of $\{1,2,...,m\}$ that the sequence could contain and yet only 2^{m-1} possible states of our algorithm's memory. Therefore there must be two different subsets S_1 and S_2 that lead to the same memory state. If S_1 and S_2 are of different sizes, then clearly this implies an error for one of the input sequences. On the other hand, if they are the same size, then if the next element is in S_1 but not S_2, the algorithm will give the same answer in both cases and therefore must give an incorrect answer on at least one of them.

Algorithm for the Number of distinct elements

To beat the above lower bound, consider approximating the number of distinct elements. Our algorithm will produce a number that is within a constant factor of the correct answer using randomization and thus a small probability of failure. First, the idea: suppose the set S of distinct elements was itself chosen uniformly at random from $\{1,...,m\}$. Let *min* denote the minimum element in S. What is the expected value of *min*? If there was one distinct element, then its expected value would be roughly $\frac{m}{2}$. If there were two distinct elements, the expected value of the minimum would be roughly

$\frac{m}{3}$. More generally, for a random set S, the expected value of the minimum is approximately $\frac{m}{|S|+1}$. See Figure 6.2. Solving $min = \frac{m}{|S|+1}$ yields $|S| = \frac{m}{min} - 1$. This suggests keeping track of the minimum element in $O(\log m)$ space and using this equation to give

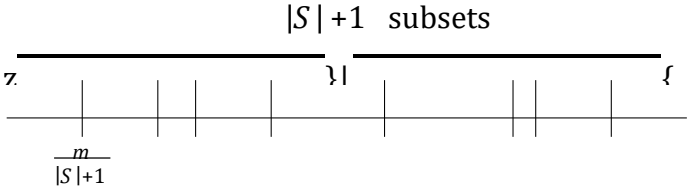


Figure 6.2: Estimating the size of S from the minimum element in S which has value approximately $\frac{m}{|S|+1}$. The elements of S partition the set $\{1,2,...,m\}$ into $|S|+1$ subsets each of size approximately $\frac{m}{|S|+1}$.

an estimate of $|S|$.

Converting the intuition into an algorithm via hashing

In general, the set S might not have been chosen uniformly at random. If the elements of S were obtained by selecting the $|S|$ smallest elements of $\{1,2,...,m\}$, the above technique would give a very bad answer. However, we can convert our intuition into an algorithm that works well with high probability on every sequence via hashing. Specifically, we will use a hash function h where

$$h : \{1,2,...,m\} \to \{0,1,2,...,M-1\},$$

and then instead of keeping track of the minimum element $a_i \in S$, we will keep track of the minimum *hash value*. The question now is: what properties of a hash function do we need? Since we need to store h, we cannot use a totally random mapping since that would take too many bits. Luckily, a pairwise independent hash function, which can be stored compactly is sufficient.

We recall the formal definition of pairwise independence below. But first recall that a hash function is always chosen at random from a family of hash functions and phrases like "probability of collision" refer to the probability in the choice of hash function.

2-Universal (Pairwise Independent) Hash Functions

A set of hash functions

$$H = \{h \mid h : \{1, 2, \ldots, m\} \to \{0, 1, 2, \ldots, M-1\}\}$$

is *2-universal* or *pairwise independent* if for all x and y in $\{1,2,...,m\}$ with $x \neq y$, $h(x)$ and $h(y)$ are each equally likely to be any element of $\{0,1,2,...,M-1\}$ and are statistically

independent. It follows that a set of hash functions H is 2-universal if and only if for all x and y in {1,2,...,m}, $x \neq y$, $h(x)$ and $h(y)$ are each equally likely to be any element of {0,1,2,...,$M-1$}, and for all w,z we have:

$$\text{Prob } h(x) = w \text{ and } h(y) = z) = \frac{1}{M^2}. \quad h \sim H$$

We now give an example of a 2-universal family of hash functions. Let M be a prime greater than m. For each pair of integers a and b in the range $[0, M-1]$, define a hash function

$$h_{ab}(x) = ax + b \pmod{M}$$

To store the hash function h_{ab}, store the two integers a and b. This requires only $O(\log M)$ space. To see that the family is 2-universal note that $h(x) = w$ and $h(y) = z$ if and only if

$$\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} w \\ z \end{pmatrix} \pmod{M}$$

If $x \neq y$, the matrix $\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}$ is invertible modulo M.[28] Thus

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}^{-1} \begin{pmatrix} w \\ z \end{pmatrix} \pmod{M}$$

and for each $\binom{w}{z}$ there is a unique $\binom{a}{b}$. Hence

$$\text{Prob } h(x) = w \text{ and } h(y) = z) = \frac{1}{M^2}$$

and H is 2-universal.

Analysis of distinct element counting algorithm

Let $b_1, b_2, \ldots, b_d$ be the distinct values that appear in the input. Select h from the 2-universal family of hash functions H. Then the set $S = \{h(b_1), h(b_2), \ldots, h(b_d)\}$ is a set of d random and pairwise independent values from the set {0,1,2,...,$M-1$}. We now show that $\frac{M}{\min}$ is a good estimate for d, the number of distinct elements in the input, where $min = min(S)$.

Lemma 6.1 *With probability at least* $\frac{2}{3} - \frac{d}{M}$, *we have* $\frac{d}{6} \leq \frac{M}{\min} \leq 6d$, *where min is the smallest element of S.*

Proof: First, we show that Prob $\frac{M}{\min} > 6d) < \frac{1}{6} + \frac{d}{M}$. This part does not require pairwise independence.

[28] The primality of M ensures that inverses of elements exist in Z_{M^*} and $M > m$ ensures that if $x \neq y$, then x and y are not equal mod M.

Prob $\left(\frac{M}{\min} > 6d\right)$ = Prob $\left(\min < \frac{M}{6d}\right)$ = Prob $\left(\exists k,\ h\left(b_k\right) < \frac{M}{6d}\right)$ $\leq \sum_{i=1}^{d}$ Prob.

$\left(h(b_i) < \frac{M}{6d}\right) \leq d\left(\frac{\lceil \frac{M}{6d} \rceil}{M}\right) \leq d\left(\frac{1}{6d} + \frac{1}{M}\right) \leq \frac{1}{6} + \frac{d}{M}$

t

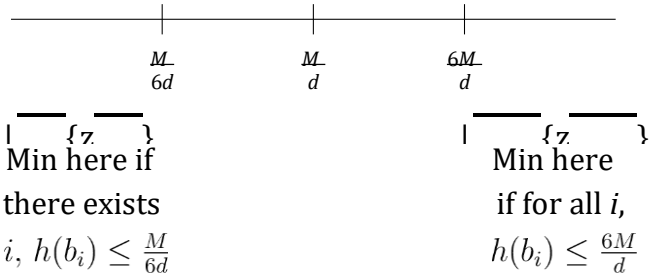


Figure 6.3: Location of the minimum in the distinct counting algorithm.

Next, we show that Prob $\left(\frac{M}{\min} < \frac{d}{6}\right) < \frac{1}{6}$. This part will use pairwise independence. First, Prob $\left(\frac{M}{\min} < \frac{d}{6}\right)$ = Prob $\left(\min > \frac{6M}{d}\right)$ = Prob $\left(\forall k,\ h\left(b_k\right) > \frac{6M}{d}\right)$. For i = 1,2,...,d, define the indicator variable

$$y_i = \begin{cases} 0 \text{ if } h(b_i) > \frac{6M}{d} \\ 1 \text{ otherwise} \end{cases}$$

and let

$$y = \sum_{i=1}^{d} y_i.$$

We want to show that with good probability, we will see a hash value in $[0, \frac{6M}{d}]$, i.e., that Prob(y = 0) is small. Now Prob(y_i = 1) ≥ $\frac{6}{d}$, $E(y_i) \geq \frac{6}{d}$, and $E(y) \geq 6$. For 2-way independent random variables, the variance of their sum is the sum of their variances. So Var(y) = dVar(y_1). Further, since y_1 is 0 or 1, Var(y_1) $= E\left[(y_1 - E(y_1))^2\right] = E(y_1^2) - E^2(y_1) = E(y_1) - E^2(y_1) \leq E(y_1)$. Thus Var($y$) ≤ $E(y)$. By the Chebyshev inequality,

Prob $\left(\frac{M}{\min} < \frac{d}{6}\right)$ = Prob $\left(\min > \frac{6M}{d}\right)$ = Prob $\left(\forall k\ h\left(b_k\right) > \frac{6M}{d}\right)$

$$
\begin{aligned}
&= \text{Prob}(y = 0) \\
&\le \text{Prob}(|y - E(y)| \ge E(y)) \\
&\le \frac{\text{Var}(y)}{E^2(y)} \le \frac{1}{E(y)} \le \frac{1}{6}
\end{aligned}
$$

Since $\frac{M}{\min} > 6d$ with probability at most $\frac{1}{6} + \frac{d}{M}$ and $\frac{M}{\min} < \frac{d}{6}$ with probability at most $\frac{1}{6}$, $\frac{d}{6} \le \frac{M}{\min} \le 6d$ with probability at least $\frac{2}{3} - \frac{d}{M}$. ■

6.2.2 Number of Occurrences of a Given Element.

To count the number of occurrences of a given element in a stream requires at most log$n$ space where n is the length of the stream. Clearly, for any length stream that occurs in practice, one can afford log$n$ space. For this reason, the following material may never be used in practice, but the technique is interesting and may give insight into how to solve some other problem.

Consider a string of 0's and 1's of length n in which we wish to count the number of occurrences of 1's. Clearly with log$n$ bits of memory we could keep track of the exact number of 1's. However, the number can be approximated with only loglog$n$ bits.

Let m be the number of 1's that occur in the sequence. Keep a value k such that 2^k is approximately the number m of occurrences. Storing k requires only loglog$n$ bits of memory. The algorithm works as follows. Start with k=0. For each occurrence of a 1, add one to k with probability $1/2^k$. At the end of the string, the quantity $2^k - 1$ is the estimate of m. To obtain a coin that comes down heads with probability $1/2^k$, flip a fair coin, one that comes down heads with probability $1/2$, k times and report heads if the fair coin comes down heads in all k flips.

Given k, on average it will take 2^k ones before k is incremented. Thus, the expected number of 1's to produce the current value of k is $1 + 2 + 4 + \cdots + 2^{k-1} = 2^k - 1$.

6.2.3 Frequent Elements

The Majority and Frequent Algorithms

First consider the very simple problem of n people voting. There are m candidates, $\{1,2,\ldots,m\}$. We want to determine if one candidate gets a majority vote and if so who. Formally, we are given a stream of integers $a_1,a_2,\ldots,a_n$, each a_i belonging to $\{1,2,\ldots,m\}$, and want to determine whether there is some $s \in \{1,2,\ldots,m\}$ which occurs more than $n/2$ times and if so which s. It is easy to see that to solve the problem exactly on read-once streaming data with a deterministic algorithm, requires $\Omega(\min(n,m))$ space. Suppose n is even and the last $n/2$ items are identical. Suppose also that after reading the first $n/2$ items, there

are two different sets of elements that result in the same content of our memory. In that case, a mistake would occur if the second half of the stream consists solely of an element that is in one set, but not in the other. If $n/2 \geq m$ then there are at least $2^m - 1$ possible subsets of the first $n/2$ elements. If $n/2 \leq m$ then there are $\sum_{i=1}^{n/2} \binom{m}{i}$ subsets. By the above argument, the number of bits of memory must be at least the base 2 logarithm of the number of subsets, which is $\Omega(\min(m,n))$.

Surprisingly, we can bypass the above lower bound by slightly weakening our goal. Again let's require that if some element appears more than $n/2$ times, then we must output it. But now, let us say that if no element appears more than $n/2$ times, then our algorithm may output whatever it wants, rather than requiring that it output "no". That is, there may be "false positives", but no "false negatives".

Majority Algorithm

> Store a_1 and initialize a counter to one. For each subsequent a_i, if a_i is the same as the currently stored item, increment the counter by one. If it differs, decrement the counter by one provided the counter is nonzero. If the counter is zero, then store a_i and set the counter to one.

To analyze the algorithm, it is convenient to view the decrement counter step as "eliminating" two items, the new one and the one that caused the last increment in the counter. It is easy to see that if there is a majority element s, it must be stored at the end. If not, each occurrence of s was eliminated; but each such elimination also causes another item to be eliminated. Thus for a majority item not to be stored at the end, more than n items must have eliminated, a contradiction.

Next we modify the above algorithm so that not just the majority, but also items with frequency above some threshold are detected. More specifically, the algorithm below finds the frequency (number of occurrences) of each element of $\{1,2,...,m\}$ to within an additive term of $\frac{n}{k+1}$. That is, for each symbol s, the algorithm produces a value $\tilde{f}_s$ in $[f_s - \frac{n}{k+1}, f_s]$, where f_s is the true number of occurrences of symbol s in the sequence. It will do so using $O(k \log n + k \log m)$ space by keeping k counters instead of just one counter.

Algorithm Frequent

> Maintain a list of items being counted. Initially the list is empty. For each item, if it is the same as some item on the list, increment its counter by one. If it differs from all the items on the list, then if there are less than k items on the list, add the item to the list with its counter set to one. If there are already k items on the list, decrement each of the current counters by one. Delete an element from the list if its count becomes zero.

Theorem 6.2 *At the end of Algorithm Frequent, for each* $s \in \{1,2,...,m\}$*, its counter on the list* $\tilde{f}_s$ *satisfies* $\tilde{f}_s \in [f_s - \frac{n}{k+1}, f_s]$*. If some s does not occur on the list, its counter is zero and the theorem asserts that* $f_s \leq \frac{n}{k+1}$.

Proof: The fact that $\tilde{f}_s \leq f_s$ is immediate. To show $\tilde{f}_s \geq f_s - \frac{n}{k+1}$, view each decrement counter step as eliminating some items. An item is eliminated if the current a_i being read is not on the list and there are already k symbols different from it on the list; in this case, a_i and k other distinct symbols are simultaneously eliminated. Thus, the elimination of each occurrence of an $s \in \{1,2,...,m\}$ is really the elimination of $k + 1$ items corresponding to distinct symbols. Thus, no more than $n/(k + 1)$ occurrences of any symbol can be eliminated. It is clear that if an item is not eliminated, then it must still be on the list at the end. This proves the theorem. ∎

Theorem 6.2 implies that we can compute the true frequency of every $s \in \{1,2,...,m\}$ to within an additive term of $\frac{n}{k+1}$.

6.2.4 The Second Moment

This section focuses on computing the second moment of a stream with symbols from $\{1,2,...,m\}$. Let f_s denote the number of occurrences of the symbol s in the stream, and recall that the second moment of the stream is given by $\sum_{s=1}^{m} f_s^2$. To calculate the second moment, for each symbol s, $1 \leq s \leq m$, independently set a random variable x_s to ±1 with probability 1/2. In particular, think of x_s as the output of a random hash function $h(s)$ whose range is just the two buckets $\{-1,1\}$. For now, think of h as a fully independent hash function. Maintain a sum by adding x_s to the sum each time the symbol s occurs in the stream. At the end of the stream, the sum will equal $\sum_{s=1}^{m} x_s f_s$. The expected value of the sum will be zero where the expectation is over the choice of the ±1 value for the x_s.

$$E\left(\sum_{s=1}^{m} x_s f_s\right) = 0$$

Although the expected value of the sum is zero, its actual value is a random variable and the expected value of the square of the sum is given by

$$E\left(\sum_{s=1}^{m} x_s f_s\right)^2 = E\left(\sum_{s=1}^{m} x_s^2 f_s^2\right) + 2E\left(\sum_{s \neq t} x_s x_t f_s f_t\right) = \sum_{s=1}^{m} f_s^2,$$

The last equality follows since $E(x_s x_t) = E(x_s)E(x_t) = 0$ for $s \neq t$, using pairwise independence of the random variables. Thus

$$a = \left(\sum_{s=1}^{m} x_s f_s\right)^2$$

is an unbiased estimator of $\sum_{s=1}^{m} f_s^2$ in that it has the correct expectation. Note that at this point we could use Markov's inequality to state that Prob($a \geq 3\sum_{s=1}^{m} f_s^2$) $\leq 1/3$, but we want to get a tighter guarantee. To do so, consider the second moment of *a*:

$$E(a^2) = E\left(\sum_{s=1}^{m} x_s f_s\right)^4 = E\left(\sum_{1\leq s,t,u,v\leq m} x_s x_t x_u x_v f_s f_t f_u f_v\right).$$

The last equality is by expansion. Assume that the random variables x_s are 4-wise independent, or equivalently that they are produced by a 4-wise independent hash function. Then, since the x_s are independent in the last sum, if any one of *s*, *u*, *t*, or *v* is distinct from the others, then the expectation of the term is zero. Thus, we need to deal only with terms of the form $x_s^2 x_t^2$ for $t \neq s$ and terms of the form x_s^4.

Each term in the above sum has four indices, *s,t,u,v,* and there are $\binom{4}{2}$ ways of choosing two indices that have the same *x* value. Thus,

$$\begin{aligned} E(a^2) &\leq \binom{4}{2} E\left(\sum_{s=1}^{m}\sum_{t=s+1}^{m} x_s^2 x_t^2 f_s^2 f_t^2\right) + E\left(\sum_{s=1}^{m} x_s^4 f_s^4\right) \\ &= 6\sum_{s=1}^{m}\sum_{t=s+1}^{m} f_s^2 f_t^2 + \sum_{s=1}^{m} f_s^4 \\ &\leq 3\left(\sum_{s=1}^{m} f_s^2\right)^2 = 3E^2(a). \end{aligned}$$

!

Therefore, $Var(a) = E(a^2) - E^2(a) \leq 2E^2(a)$.

Since the variance is comparable to the square of the expectation, repeating the process several times and taking the average, gives high accuracy with high probability.

Theorem 6.3 *The average* x *of* $r = \frac{2}{\varepsilon^2\delta}$ *estimates* $a_1,\ldots,a_r$ *using independent sets of 4-way independent random variables is*

$$Prob(|x - E(x)| > \varepsilon E(x)) < \frac{Var(x)}{\epsilon^2 E^2(x)} \leq \delta.$$

Proof: The proof follows from the fact that taking the average of *r* independent repetitions reduces variance by a factor of *r*, so that $Var(x) \leq \delta\varepsilon^2 E^2(x)$, and then applying Chebyshev's inequality. ∎

It remains to show that we can implement the desired 4-way independent random variables using $O(\log m)$ space. We earlier gave a construction for a pairwise-independent

set of hash functions; now we need 4-wise independence, though only into a range of {−1,1}. Below we present one such construction.

Error-Correcting codes, polynomial interpolation and limited-way independence

Consider the problem of generating a random m-dimensional vector $\mathbf{x}$ of ±1's so that any four coordinates are mutually independent. Such an m-dimensional vector may be generated from a truly random "seed" of only $O(\log m)$ mutually independent bits. Thus, we need only store the $O(\log m)$ bits and can generate any of the m coordinates when needed. For any k, there is a finite field F with exactly 2^k elements, each of which can be represented with k bits and arithmetic operations in the field can be carried out in $O(k^2)$ time. Here, k is the ceiling of $\log_2 m$. A basic fact about polynomial interpolation is that a polynomial of degree at most three is uniquely determined by its value over any field F at four points. More precisely, for any four distinct points a_1,a_2,a_3,a_4 in F and any four possibly not distinct values b_1,b_2,b_3,b_4 in F, there is a unique polynomial $f(x) = f_0 + f_1x + f_2x^2 + f_3x^3$ of degree at most three, so that with computations done over F, $f(a_i) = b_i$ $1 \le i \le 4$.

The definition of the pseudo-random ±1 vector $\mathbf{x}$ with 4-way independence is simple. Choose four elements f_0,f_1,f_2,f_3 at random from F and form the polynomial $f(s) = f_0 + f_1s + f_2s^2 + f_3s^3$. This polynomial represents $\mathbf{x}$ as follows. For $s = 1,2,\ldots,m$, x_s is the leading bit of the k-bit representation of $f(s)$.[29] Thus, the m-dimensional vector $\mathbf{x}$ requires only $O(k)$ bits where $k = \text{dlog}m\text{e}$.

Lemma 6.4 *The* $\mathbf{x}$ *defined above has 4-way independence.*

Proof: Assume that the elements of F are represented in binary using ±1 instead of the traditional 0 and 1. Let s, t, u, and v be any four coordinates of $\mathbf{x}$ and let $\alpha,\beta,\gamma,$ and δ have values in ±1. There are exactly 2^{k-1} elements of F whose leading bit is α and similarly for β, γ, and δ. So, there are exactly $2^{4(k-1)}$ 4-tuples of elements b_1, b_2, b_3, and b_4 in F so that the leading bit of b_1 is α, the leading bit of b_2 is β, the leading bit of b_3 is γ, and the leading bit of b_4 is δ. For each such b_1, b_2, b_3, and b_4, there is precisely one polynomial f so that $f(s) = b_1$, $f(t) = b_2$, $f(u) = b_3$, and $f(v) = b_4$. The probability that $x_s = \alpha$, $x_t = \beta$, $x_u = \gamma$, and $x_v = \delta$ is precisely

$$\frac{2^{4(k-1)}}{\text{total number of } f} = \frac{2^{4(k-1)}}{2^{4k}} = \frac{1}{16}.$$

Four way independence follows since $\text{Prob}(x_s = \alpha) = \text{Prob}(x_t = \beta) = \text{Prob}(x_u = \gamma) =$

[29] Here we have numbered the elements of the field F $s = 1,2,\ldots,m$.

Prob($x_v = \delta$) = 1/2 and thus

$$\text{Prob}(x_s = \alpha)\text{Prob}(x_t = \beta)\text{Prob}(x_u = \gamma)\text{Prob}(x_v = \delta)$$
$$= \text{Prob}(x_s = \alpha,\ x_t = \beta,\ x_u = \gamma \text{ and } x_s = \delta)$$ ∎

Lemma 6.4 describes how to get one vector **x** with 4-way independence. However, we need $r = O(1/\varepsilon^2)$ mutually independent vectors. Choose r independent polynomials at the outset.

To implement the algorithm with low space, store only the polynomials in memory. This requires $4k = O(\log m)$ bits per polynomial for a total of $O(\frac{\log m}{\varepsilon^2})$ bits. When a symbol s in the stream is read, compute each polynomial at s to obtain the value for the corresponding value of the x_s and update the running sums. x_s is just the leading bit of the value of the polynomial evaluated at s. This calculation requires $O(\log m)$ time. Thus, we repeatedly compute the x_s from the "seeds", namely the coefficients of the polynomials.

This idea of polynomial interpolation is also used in other contexts. Error-correcting codes is an important example. To transmit n bits over a channel which may introduce noise, one can introduce redundancy into the transmission so that some channel errors can be corrected. A simple way to do this is to view the n bits to be transmitted as coefficients of a polynomial $f(x)$ of degree $n - 1$. Now transmit f evaluated at points 1,2,3,...,$n + m$. At the receiving end, any n correct values will suffice to reconstruct the polynomial and the true message. So up to m errors can be tolerated. But even if the number of errors is at most m, it is not a simple matter to know which values are corrupted. We do not elaborate on this here.

6.3 Matrix Algorithms using Sampling

We now move from the streaming model to a model where the input is stored in memory, but because the input is so large, one would like to produce a much smaller approximation to it, or perform an approximate computation on it in low space. For instance, the input might be stored in a large slow memory and we would like a small "sketch" that can be stored in smaller fast memory and yet retains the important properties of the original input. In fact, one can view a number of results from the chapter on machine learning in this way: we have a large population, and we want to take a small sample, perform some optimization on the sample, and then argue that the optimum solution on the sample will be approximately optimal over the whole population. In the chapter on machine learning, our sample consisted of independent random draws from the overall population or data distribution. Here we will be looking at matrix algorithms and to achieve errors that are small compared to the Frobenius norm of the matrix rather than compared to the total number of entries, we will perform non-uniform sampling.

Algorithms for matrix problems like matrix multiplication, low-rank approximations, singular value decomposition, compressed representations of matrices, linear regression etc. are widely used but some require $O(n^3)$ time for $n \times n$ matrices.

The natural alternative to working on the whole input matrix is to pick a random sub-matrix and compute with that. Here, we will pick a subset of columns or rows of the input matrix. If the sample size s is the number of columns we are willing to work with, we will do s independent identical trials. In each trial, we select a column of the matrix. All that we have to decide is what the probability of picking each column is. Sampling uniformly at random is one option, but it is not always good if we want our error to be a small fraction of the Frobenius norm of the matrix. For example, suppose the input matrix has all entries in the range [−1,1] but most columns are close to the zero vector with only a few significant columns. Then, uniformly sampling a small number of columns is unlikely to pick up any of the significant columns and essentially will approximate the original matrix with the all-zeroes matrix.[30]

We will see that the "optimal" probabilities are proportional to the squared length of columns. This is referred to as length squared sampling and since its first discovery in the mid-90's, has been proved to have several desirable properties which we will see. Note that all sampling we will discuss here is done with replacement.

Two general notes on this approach:

(i) We will prove error bounds which hold for all input matrices. Our algorithmsare randomized, i.e., use a random number generator, so the error bounds are random variables. The bounds are on the expected error or tail probability bounds on large errors and apply to any matrix. Note that this contrasts with the situation where we have a stochastic model of the input matrix and only assert error bounds for "most" matrices drawn from the probability distribution of the stochastic model. A mnemonic is - our algorithms can toss coins, but our data does not toss coins. A reason for proving error bounds for any matrix is that in real problems, like the analysis of the web hypertext link matrix or the patient-genome expression matrix, it is the one matrix the user is interested in, not a random matrix. In general, we focus on general algorithms and theorems, not specific applications, so the reader need not be aware of what the two matrices above mean.

(ii) There is "no free lunch". Since we only work on a small random sample and noton the whole input matrix, our error bounds will not be good for certain matrices. For

[30] There are, on the other hand, many positive statements one *can* make about uniform sampling. For example, suppose the columns of A are data points in an m-dimensional space (one dimension per row). Fix any k-dimensional subspace, such as the subspace spanned by the k top singular vectors. If we randomly sample$\tilde{O}(k/\epsilon^2)$ columns uniformly, by the VC-dimension bounds given in Chapter 6, with high probability for every vector $\mathbf{v}$ in the k-dimensional space and every threshold τ, the fraction of the

example, if the input matrix is the identity, it is intuitively clear that picking a few random columns will miss the other directions.

To the Reader: Why aren't (i) and (ii) mutually contradictory?

6.3.1 Matrix Multiplication using Sampling

Suppose A is an $m \times n$ matrix and B is an $n \times p$ matrix and the product AB is desired. We show how to use sampling to get an approximate product faster than the traditional multiplication. Let $A(:,k)$ denote the k^{th} column of A. $A(:,k)$ is a $m \times 1$ matrix. Let $B(k,:)$ be the k^{th} row of B. $B(k,:)$ is a $1 \times n$ matrix. It is easy to see that

$$AB = \sum_{k=1}^{n} A(:,k)B(k,:)$$

Note that for each value of k, $A(:,k)B(k,:)$ is an $m \times p$ matrix each element of which is a single product of elements of A and B. An obvious use of sampling suggests itself. Sample some values for k and compute $A(:,k)B(k,:)$ for the sampled k's and use their suitably scaled sum as the estimate of AB. It turns out that nonuniform sampling probabilities are useful. Define a random variable z that takes on values in $\{1,2,...,n\}$. Let p_k denote the probability that z assumes the value k. We will solve for a good choice of probabilities

---

sampled columns $\mathbf{a}$ that satisfy $\mathbf{v}^T\mathbf{a} \geq \tau$ will be within $\pm$ of the fraction of the columns $\mathbf{a}$ in the overall matrix A satisfying $\mathbf{v}^T\mathbf{a} \geq \tau$.

later, but for now just consider the p_k as nonnegative numbers that sum to one. Define an associated random matrix variable that has value

$$X = \frac{1}{p_k} A(:,k)\, B(k,:) \qquad (6.1)$$

with probability p_k. Let $E(X)$ denote the entry-wise expectation.

$$E(X) = \sum_{k=1}^{n} \text{Prob}(z = k)\frac{1}{p_k} A(:,k)\, B(k,:) = \sum_{k} A(:,k)B(k,:) = AB.$$

This explains the scaling by $\frac{1}{p_k}$ in X. In particular, X is a matrix-valued random variable each of whose components is correct in expectation. We will be interested in

$$E\left(||AB - X||_F^2\right).$$

This can be viewed as the variance of X, defined as the sum of the variances of all its entries.

$$\mathrm{Var}(X) = \sum_{i=1}^{m}\sum_{j=1}^{p}\mathrm{Var}\left(x_{ij}\right) = \sum_{ij} E\left(x_{ij}^2\right) - E\left(x_{ij}\right)^2 = \left(\sum_{ij}\sum_{k} p_k \frac{1}{p_k^2} a_{ik}^2 b_{kj}^2\right) - ||AB||_F^2 .$$

We want to choose p_k to minimize this quantity, and notice that we can ignore the $||AB||^2_F$ term since it doesn't depend on the p_k's at all. We can now simplify by exchanging the order of summations to get

$$\sum_{ij}\sum_{k} p_k \frac{1}{p_k^2} a_{ik}^2 b_{kj}^2 = \sum_{k} \frac{1}{p_k}\left(\sum_i a_{ik}^2\right)\left(\sum_j b_{kj}^2\right) = \sum_k \frac{1}{p_k}\left|A\left(:,k\right)\right|^2\left|B\left(k,:\right)\right|^2 .$$

What is the best choice of p_k to minimize this sum? It can be seen by calculus[31] that the minimizing p_k are proportional to $|A(:,k)||B(k,:)|$. In the important special case when $B = A^T$, pick columns of A with probabilities proportional to the squared length of the columns. Even in the general case when B is not A^T, doing so simplifies the bounds.

This sampling is called "length squared sampling". If p_k is proportional to $|A(:,k)|^2$, i.e, $p_k = \frac{|A(:,k)|^2}{||A||_F^2}$, then

$$E\left(||AB - X||_F^2\right) = \mathrm{Var}(X) \leq ||A||_F^2 \sum_k |B\left(k,:\right)|^2 = ||A||_F^2 ||B||_F^2 .$$

To reduce the variance, we can do s independent trials. Each trial i, i = 1,2,...,s yields a matrix X_i as in (6.1). We take $\frac{1}{s}\sum_{i=1}^{s} X_i$ as our estimate of AB. Since the variance of a sum of independent random variables is the sum of variances, the variance

Corresponding

Sampled scaled rows of $s \times p$ B

Scaled

A columns $\approx$

of

[31] By taking derivatives, for any set of nonnegative numbers c_k, $\sum_k \frac{c_k}{p_k}$ is minimized with p_k proportional to $\sqrt{c_k}$.

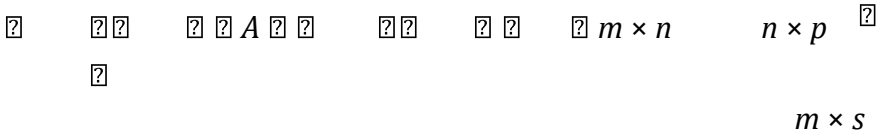


Figure 6.4: Approximate Matrix Multiplication using sampling

of $\frac{1}{s}\sum_{i=1}^{s} X_i$ is $\frac{1}{s}\mathrm{Var}(X)$ and so is at most $\frac{1}{s}||A||_F^2||B||_F^2$. Let $k_1,\ldots,k_s$ be the k's chosen in each trial. Expanding this, gives:

$$\frac{1}{s}\sum_{i=1}^{s} X_i = \frac{1}{s}\left(\frac{A\left(:,k_1\right)B\left(k_1,:\right)}{p_{k_1}} + \frac{A\left(:,k_2\right)B\left(k_2,:\right)}{p_{k_2}} + \cdots + \frac{A\left(:,k_s\right)B\left(k_s,:\right)}{p_{k_s}}\right). \quad (6.2)$$

We will find it convieneint to write this as the product of an $m \times s$ matrix with a $s \times p$ matrix as follows: Let C be the $m \times s$ matrix consisting of the following columns which are scaled versions of the chosen columns of A:

$$\frac{A(:,k_1)}{\sqrt{sp_{k_1}}}, \frac{A(:,k_2)}{\sqrt{sp_{k_2}}}, \cdots \frac{A(:,k_s)}{\sqrt{sp_{k_s}}}.$$

Note that the scaling has a nice property, which the reader can verify:

$$E\left(CC^T\right) = AA^T. \quad (6.3)$$

Define R to be the $s \times p$ matrix with the corresponding rows of B similarly scaled, namely, R has rows

$$\frac{B(k_1,:)}{\sqrt{sp_{k_1}}}, \frac{B(k_2,:)}{\sqrt{sp_{k_2}}}, \cdots \frac{B(k_s,:)}{\sqrt{sp_{k_s}}}.$$

The reader may verify that

$$E\left(R^T R\right) = B^T B. \quad (6.4)$$

From (6.2), we see that $\frac{1}{s}\sum_{i=1}^{s} X_i = CR$. This is represented in Figure 6.4. We summarize our discussion in Theorem 6.3.1.

Theorem 6.5 *Suppose A is an m × n matrix and B is an n × p matrix. The product AB can be estimated by CR, where C is an m × s matrix consisting of s columns of A picked according to length-squared distribution and scaled to satisfy (6.3) and R is the s × p matrix consisting of the corresponding rows of B scaled to satisfy (6.4). The error is bounded by:*

$$E\left(||AB - CR||_F^2\right) \leq \frac{||A||_F^2\ ||B||_F^2}{s}.$$

Thus, to ensure E $(||AB - CR||^2_F) \leq \varepsilon^2||A||^2_F||B||^2_F$*, it suffices to make s greater than or equal to* $1/\varepsilon^2$*. If ε is* $\Omega(1)$*, so* $s \in O(1)$*, then the multiplication CR can be carried out in time* $O(mp)$*.*

When is this error bound good and when is it not? Let's focus on the case that $B = A^T$ so we have just one matrix to consider. If A is the identity matrix, then the guarantee is not very good. In this case, $||AA^T||_F^2 = n$, but the right-hand-side of the inequality is $\frac{n^2}{s}$. So we would need $s > n$ for the bound to be any better than approximating the product with the zero matrix.

More generally, the trivial estimate of the all zero matrix for AA^T makes an error in Frobenius norm of $||AA^T||_F$. What s do we need to ensure that the error is at most this? If $\sigma_1, \sigma_2, \ldots$ are the singular values of A, then the singular values of AA^T are $\sigma_1^2, \sigma_2^2, \cdots$ and

$$||AA^T||_F^2 = \sum_t \sigma_t^4 \qquad \text{and} \qquad ||A||_F^2 = \sum_t \sigma_t^2.$$

So from Theorem 6.3.1, $E(||AA^T - CR||_F^2) \leq ||AA^T||_F^2$ provided

$$s \geq \frac{(\sigma_1^2 + \sigma_2^2 + \ldots)^2}{\sigma_1^4 + \sigma_2^4 + \ldots}.$$

If rank(A) = r, then there are r non-zero σ_t and the best general upper bound on the ratio $\frac{(\sigma_1^2+\sigma_2^2+\ldots)^2}{\sigma_1^4+\sigma_2^4+\ldots}$ is r, so in general, s needs to be at least r. If A is full rank, this means sampling will not gain us anything over taking the whole matrix!

However, if there is a constant c and a small integer p such that

$$\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_p^2 \geq c(\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2), \tag{6.5}$$

then,

$$\frac{(\sigma_1^2 + \sigma_2^2 + \ldots)^2}{\sigma_1^4 + \sigma_2^4 + \ldots} \leq c^2 \frac{(\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_p^2)^2}{\sigma_1^4 + \sigma_2^4 + \ldots + \sigma_p^2} \leq c^2 p,$$

and so $s \geq c^2 p$ gives us a better estimate than the zero matrix. Increasing s by a factor decreases the error by the same factor. Condition 6.5 is indeed the hypothesis of the subject of Principal Component Analysis (PCA) and there are many situations when the data matrix does satisfy the condition and sampling algorithms are useful.

6.3.2 Implementing Length Squared Sampling in Two Passes

Traditional matrix algorithms often assume that the input matrix is in random access memory (RAM) and so any particular entry of the matrix can be accessed in unit time. For massive matrices, RAM may be too small to hold the entire matrix, but may be able to hold and compute with the sampled columns and rows.

Consider a high-level model where the input matrix or matrices have to be read from external memory using one pass in which one can read sequentially all entries of the matrix and sample.

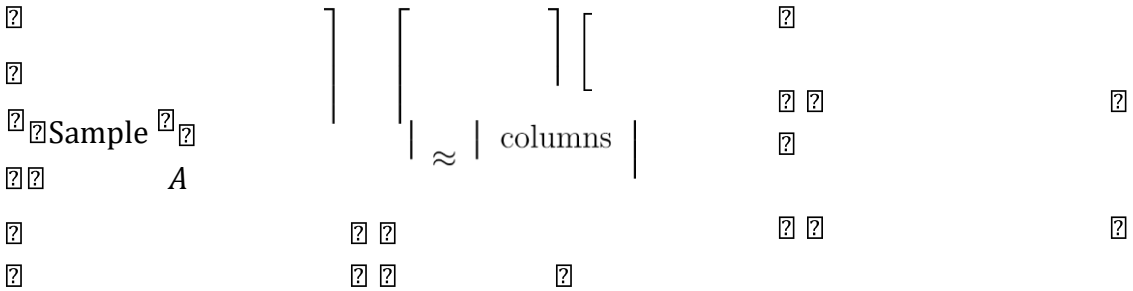
It is easy to see that two passes suffice to draw a sample of columns of A according to length squared probabilities, even if the matrix is not in row-order or column-order and entries are presented as a linked list. In the first pass, compute the length squared of each column and store this information in RAM. The lengths squared can be computed as running sums. Then, use a random number generator in RAM to determine according to length squared probability the columns to be sampled. Then, make a second pass picking the columns to be sampled.

If the matrix is already presented in external memory in column-order, then one pass will do. The idea is to use the primitive in Section 6.1: given a read-once stream of positive numbers $a_1,a_2,...,a_n$, at the end have an $i \in \{1,2,...,n\}$ such that the probability that i was chosen is $\frac{a_i}{\sum_{j=1}^{n} a_j}$. Filling in the specifics is left as an exercise for the reader.

6.3.3 Sketch of a Large Matrix

The main result of this section is that for any matrix, a sample of columns and rows, each picked according to length squared distribution provides a good sketch of the matrix. Let A be an $m \times n$ matrix. Pick s columns of A according to length squared distribution. Let C be the $m \times s$ matrix containing the picked columns scaled so as to satisy (6.3), i.e., if $A(:,k)$ is picked, it is scaled by $1/\sqrt{sp_k}$. Similarly, pick r rows of A according to length squared distribution on the rows of A. Let R be the $r \times n$ matrix of the picked rows, scaled as follows: If row k of A is picked, it is scaled by $1/\sqrt{rp_k}$. We then have $E(R^T R) = A^T A$. From C and R, one can find a matrix U so that $A \approx CUR$. The schematic diagram is given in Figure 6.5.

One may recall that the top k singular vectors of the SVD of A give a similar picture; however, the SVD takes more time to compute, requires all of A to be stored in RAM, and does not have the property that the rows and columns are directly from A. The last property, that the approximation involves actual rows/columns of the matrix rather than linear combinations, is called an *interpolative approximation* and is useful in many contexts. On the other hand, the SVD yields the best 2-norm approximation. Error bounds for the approximation CUR are weaker.



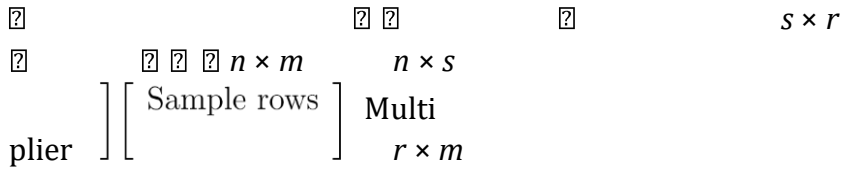


Figure 6.5: Schematic diagram of the approximation of A by a sample of s columns and r rows.

We briefly touch upon two motivations for such a sketch. Suppose A is the documentterm matrix of a large collection of documents. We are to "read" the collection at the outset and store a sketch so that later, when a query represented by a vector with one entry per term arrives, we can find its similarity to each document in the collection. Similarity is defined by the dot product. In Figure 6.5 it is clear that the matrix-vector product of a query with the right hand side can be done in time $O(ns + sr + rm)$ which would be linear in n and m if s and r are $O(1)$. To bound errors for this process, we need to show that the difference between A and the sketch of A has small 2-norm. Recall that the 2-norm $||A||_2$ of a matrix A is $\max_{|\mathbf{x}|=1} |A\mathbf{x}|$. The fact that the sketch is an interpolative approximation means that our approximation essentially consists a subset of documents and a subset of terms, which may be thought of as a representative set of documents and terms. Additionally, if A is sparse in its rows and columns, each document contains only a small fraction of the terms and each term is in only a small fraction of the documents, then this sparsity property will be preserved in C and R, unlike with SVD.

A second motivation comes from analyzing gene microarray data. Here, A is a matrix in which each row is a gene and each column is a condition. Entry (i,j) indicates the extent to which gene i is expressed in condition j. In this context, a *CUR* decomposition provides a sketch of the matrix A in which rows and columns correspond to actual genes and conditions, respectively. This can often be easier for biologists to interpret than a singular value decomposition in which rows and columns would be linear combinations of the genes and conditions.

It remains now to describe how to find U from C and R. There is a $n \times n$ matrix P of the form $P = QR$ that acts as the identity on the space spanned by the rows of R and zeros out all vectors orthogonal to this space. We state this now and postpone the proof.

Lemma 6.6 *If RR^T is invertible, then $P = R^T(RR^T)^{-1}R$ has the following properties:* **(i)** *It acts as the identity matrix on the row space of R. I.e., $P\mathbf{x} = \mathbf{x}$ for every vector $\mathbf{x}$ of the form $\mathbf{x} = R^T\mathbf{y}$ (this defines the row space of R). Furthermore,*

(ii) *if $\mathbf{x}$ is orthogonal to the row space of R, then $P\mathbf{x} = \mathbf{0}$.*

If RR^T is not invertible, let rank $(RR^T) = r$ *and* $RR^T = \sum_{t=1}^{r} \sigma_t \mathbf{u_t}\mathbf{v_t}^T$ *be the SVD of* RR^T*. Then,*

$$P = R^T \left(\sum_{t=1}^{r} \frac{1}{\sigma_t^2} \mathbf{u_t}\mathbf{v_t}^T \right) R$$

satisfies (i) and (ii).

We begin with some intuition. In particular, we first present a simpler idea that does not work, but that motivates an idea that does. Write A as AI, where I is the $n \times n$ identity matrix. Approximate the product AI using the algorithm of Theorem 6.3.1, i.e., by sampling s columns of A according to a length-squared distribution. Then, as in the last section, write $AI \approx CW$, where W consists of a scaled version of the s rows of I corresponding to the s columns of A that were picked. Theorem 6.3.1 bounds the error $||A-CW||_F^2$ by $||A||_F^2||I||_F^2/s = \frac{n}{s}||A||_F^2$. But we would like the error to be a small fraction of $||A||_F^2$ which would require $s \geq n$, which clearly is of no use since this would pick as many or more columns than the whole of A.

Let's use the identity-like matrix P instead of I in the above discussion. Using the fact that R is picked according to length squared sampling, we will show the following proposition later.

Proposition 6.7 *$A \approx AP$ and the error $E\left(||A - AP||_2^2\right)$ is at most $\frac{1}{\sqrt{r}}||A||_F^2$.*

We then use Theorem 6.3.1 to argue that instead of doing the multiplication AP, we can use the sampled columns of A and the corresponding rows of P. The s sampled columns of A form C. We have to take the corresponding s rows of $P = R^T(RR^T)^{-1}R$, which is the same as taking the corresponding s rows of R^T, and multiplying this by $(RR^T)^{-1}R$. It is easy to check that this leads to an expression of the form CUR. Further, by Theorem 6.3.1, the error is bounded by

$$E\left(||AP - CUR||_2^2\right) \leq E\left(||AP - CUR||_F^2\right) \leq \frac{||A||_F^2||P||_F^2}{s} \leq \frac{r}{s}||A||_F^2, \qquad (6.6)$$

since we will show later that: **Proposition**

6.8 $||P||_F^2 \leq r$.

Putting (6.6) and Proposition 6.7 together, and using the fact that by triangle inequality $||A-CUR||_2 \leq ||A-AP||_2 + ||AP - CUR||_2$, which in turn implies that $||A - CUR||_2^2 \leq 2||A - AP||_2^2 + 2||AP - CUR||_2^2$, the main result below follows.

Theorem 6.9 *Let A be an $m \times n$ matrix and r and s be positive integers. Let C be an $m \times s$ matrix of s columns of A picked according to length squared sampling and let R be a matrix of r rows of A picked according to length squared sampling. Then, we can find from C and R an $s \times r$ matrix U so that*

$$E\left(||A-CUR||_2^2\right) \leq ||A||_F^2 \left(\frac{2}{\sqrt{r}} + \frac{2r}{s}\right).$$

If s is fixed, the error is minimized when $r = s^{2/3}$. Choosing $s = 1/\varepsilon^3$ and $r = 1/\varepsilon^2$, the bound becomes $O(\varepsilon)||A||_F^2$. When is this bound meaningful? We discuss this further after first proving all the claims used in the discussion above.

Proof of Lemma 6.6: First consider the case that RR^T is invertible. For $\mathbf{x} = R^T\mathbf{y}$, $R^T(RR^T)^{-1}R\mathbf{x} = R^T(RR^T)^{-1}RR^T\mathbf{y} = R^T\mathbf{y} = \mathbf{x}$. If $\mathbf{x}$ is orthogonal to every row of R, then $R\mathbf{x} = \mathbf{0}$, so $P\mathbf{x} = \mathbf{0}$. More generally, if $RR^T = \sum_t \sigma_t \mathbf{u_t}\mathbf{v_t}^T$, then, $R^T \sum_t \frac{1}{\sigma_t^2} R = \sum_t \mathbf{v_t}\mathbf{v_t}^T$ and clearly satisfies (i) and (ii). ∎

Next we prove Proposition 6.7. First, recall that

$$||A-AP||_2^2 = \max_{\{\mathbf{x}:|\mathbf{x}|=1\}} |(A-AP)\mathbf{x}|^2.$$

Now suppose $\mathbf{x}$ is in the row space V of R. From Lemma 6.6, $P\mathbf{x} = \mathbf{x}$, so for $\mathbf{x} \in V$, $(A-AP)\mathbf{x} = \mathbf{0}$. Since every vector can be written as a sum of a vector in V plus a vector orthogonal to V, this implies that the maximum must therefore occur at some $\mathbf{x} \in V^{\perp}$. For such $\mathbf{x}$, by Lemma 6.6, $(A-AP)\mathbf{x} = A\mathbf{x}$. Thus, the question becomes: for unit-length $\mathbf{x} \in V^{\perp}$, how large can $|A\mathbf{x}|^2$ be? To analyze this, write:

$$|A\mathbf{x}|^2 = \mathbf{x}^T A^T A\mathbf{x} = \mathbf{x}^T(A^TA - R^TR)\mathbf{x} \leq ||A^TA - R^TR||_2 |\mathbf{x}|^2 \leq ||A^TA - R^TR||_2.$$

This implies that $||A-AP||_2^2 \leq ||A^TA - R^TR||_2$. So, it suffices to prove that $||A^TA - R^TR||_2^2 \leq ||A||_F^4/r$ which follows directly from Theorem 6.3.1, since we can think of R^TR as a way of estimating A^TA by picking according to length-squared distribution columns of A^T, i.e., rows of A. This proves Proposition 6.7.

Proposition 6.8 is easy to see. By Lemma 6.6, P is the identity on the space V spanned by the rows of R, and $P\mathbf{x} = 0$ for $\mathbf{x}$ perpendicular to the rows of R. Thus $||P||_F^2$ is the sum of its singular values squared which is at most r as claimed.

We now briefly look at the time needed to compute U. The only involved step in computing U is to find $(RR^T)^{-1}$ or do the SVD of RR^T. But note that RR^T is an $r \times r$ matrix and since r is much smaller than n and m, this is fast.

Understanding the bound in Theorem 6.9: To better understand the bound in Theorem 6.9 consider when it is meaningful and when it is not. First, choose parameters $s = \Theta(1/\varepsilon^3)$ and $r = \Theta(1/\varepsilon^2)$ so that the bound becomes $E(||A-CUR||_2^2) \leq \varepsilon||A||_F^2$.

Recall that $||A||_F^2 = \sum_i \sigma_i^2(A)$, i.e., the sum of squares of all the singular values of A. Also, for convenience scale A so that $\sigma_1^2(A) = 1$. Then

$$\sigma_1^2(A) = ||A||_2^2 = 1 \quad \text{and} \quad E(||A-CUR||_2^2) \leq \varepsilon \sum_i \sigma_i^2(A).$$

i

This, gives an intuitive sense of when the guarantee is good and when it is not. If the top k singular values of A are all $\Omega(1)$ for $k \gg m^{1/3}$, so that $\sum_i \sigma_i^2(A) \gg m^{1/3}$, then the guarantee is only meaningful when $\varepsilon = o(m^{-1/3})$, which is not interesting because it requires $s > m$. On the other hand, if just the first few singular values of A are large and the rest are quite small, e.g, A represents a collection of points that lie very close to a low-dimensional subspace, and in particular if $\sum_i \sigma_i^2(A)$ is a constant, then to be meaningful the bound requires ε to be a small constant. In this case, the guarantee is indeed meaningful because it implies that a constant number of rows and columns provides a good 2-norm approximation to A.

6.4 Sketches of Documents

Suppose one wished to store all the web pages from the World Wide Web. Since there are billions of web pages, one might want to store just a sketch of each page where a sketch is some type of compact description that captures sufficient information to do whatever task one has in mind. For the current discussion, we will think of a web page as a string of characters, and the task at hand will be one of estimating similarities between pairs of web pages.

We begin this section by showing how to estimate similarities between sets via sampling, and then how to convert the problem of estimating similarities between strings into a problem of estimating similarities between sets.

Consider subsets of size 1000 of the integers from 1 to 10^6. Suppose one wished to compute the resemblance of two subsets A and B by the formula

$$\text{resemblance}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Suppose that instead of using the sets A and B, one sampled the sets and compared random subsets of size ten. How accurate would the estimate be? One way to sample would be to select ten elements uniformly at random from A and B. Suppose A and B were each of size 1000, over lapped by 500, and both were represented by six samples. Even though half of the six samples of A were in B they would not likely be among the samples representing B. See Figure 6.6. This method is unlikely to produce overlapping samples. Another way would be to select the ten smallest elements from each of A and B. If the sets A and B overlapped significantly one might expect the sets of ten smallest B

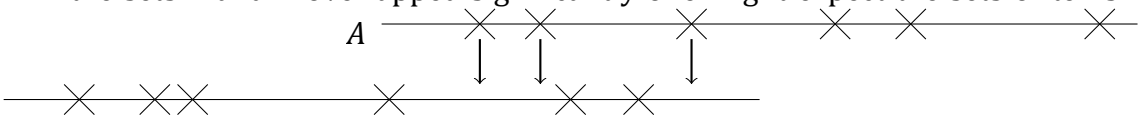


Figure 6.6: Samples of overlapping sets A and B.

elements from each of *A* and *B* to also overlap. One difficulty that might arise is that the small integers might be used for some special purpose and appear in essentially all sets and thus distort the results. To overcome this potential problem, rename all elements using a random permutation.

Suppose two subsets of size 1000 overlapped by 900 elements. What might one expect the overlap of the 10 smallest elements from each subset to be? One would expect the nine smallest elements from the 900 common elements to be in each of the two sampled subsets for an overlap of 90%. The expected resemblance(*A,B*) for the size ten sample would be 9/11=0.81.

Another method would be to select the elements equal to zero mod *m* for some integer *m*. If one samples mod *m* the size of the sample becomes a function of *n*. Sampling mod *m* allows one to handle containment.

In another version of the problem one has a string of characters rather than a set. Here one converts the string into a set by replacing it by the set of all of its substrings of some small length *k*. Corresponding to each string is a set of length *k* substrings. If *k* is modestly large, then two strings are highly unlikely to give rise to the same set of substrings. Thus, we have converted the problem of sampling a string to that of sampling a set. Instead of storing all the substrings of length *k*, we need only store a small subset of the length *k* substrings.

Suppose you wish to be able to determine if two web pages are minor modifications of one another or to determine if one is a fragment of the other. Extract the sequence of words occurring on the page, viewing each word as a character. Then define the set of substrings of *k* consecutive words from the sequence. Let *S*(*D*) be the set of all substrings of *k* consecutive words occurring in document *D*. Define resemblance of *A* and *B* by

$$\text{resemblance}(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

And define containment as

$$\text{containment}(A, B) = \frac{|S(A) \cap S(B)|}{|S(A)|}$$

Let π be a random permutation of all length *k* substrings. Define *F*(*A*) to be the *s* smallest elements of *A* and *V* (*A*) to be the set mod *m* in the ordering defined by the permutation.

Then

$$\frac{F(A) \cap F(B)}{F(A) \cup F(B)}$$

and

$$\frac{|V(A) \cap V(B)|}{|V(A) \cup V(B)|}$$

are unbiased estimates of the resemblance of *A* and *B*. The value

$$\frac{|V(A) \cap V(B)|}{|V(A)|}$$

is an unbiased estimate of the containment of A in B.

6.5 Bibliographic Notes

The hashing-based algorithm for counting the number of distrinct elements in a data stream described in Section 6.2.1 is due to Flajolet and Martin [FM85]. Algorithm Frequent for identifying the most frequent elements is due to Misra and Gries [MG82]. The algorithm for estimating the second moment of a data stream described in Section 6.2.4 is due to Alon, Matias and Szegedy [AMS96], who also gave algorithms and lower bounds for other k^{th} moments. These early algorithms for streaming data significantly influenced further research in the area. Improvements and generalizations of Algorithm Frequent were made in [MM02].

Length-squared sampling was introduced by Frieze, Kannan and Vempala [FKV04]; the algorithms of Section 6.3 are from [DKM06a, DKM06b]. The material in Section 6.4 on sketches of documents is from Broder et al. [BGMZ97].

6.6 Exercises

Exercise 6.1 *Let $a_1,a_2,\ldots,a_n$, be a stream of symbols each an integer in $\{1,\ldots,m\}$.*

1. *Give an algorithm that will select a symbol uniformly at random from the stream. How much memory does your algorithm require?*

2. *Give an algorithm that will select a symbol with probability proportional to a^2_i.*

Exercise 6.2 *How would one pick a random word from a very large book where the probability of picking a word is proportional to the number of occurrences of the word in the book?*

Exercise 6.3 *Consider a matrix where each element has a probability of being selected. Can you select a row according to the sum of probabilities of elements in that row by just selecting an element according to its probability and selecting the row that the element is in?*

Exercise 6.4 *For the streaming model give an algorithm to draw t independent samples of indices i, each with probability proportional to the value of a_i. Some images may be drawn multiple times. What is its memory usage?*

Exercise 6.5 *For some constant $c > 0$, it is possible to create 2^{cm} subsets of $\{1,\ldots,m\}$, each with $m/2$ elements, such that no two of the subsets share more than $3m/8$ elements in common.*[32] *Use this fact to argue that any deterministic algorithm that even guarantees to* approximate

[32] For example, choosing them randomly will work with high probability. You expect two subsets of size $m/2$ to share $m/4$ elements in common, and with high probability they will share no more than $3m/8$.

the number of distinct elements in a data stream with error less than $\frac{m}{16}$ *must use* $\Omega(m)$ *bits of memory on some input sequence of length* $n \leq 2m$.

Exercise 6.6 *Consider an algorithm that uses a random hash function and gives an estimate* $\hat{x}$ *of the true value x of some variable. Suppose that* $\frac{x}{4} \leq \hat{x} \leq 4x$ *with probability at least 0.6. The probability of the estimate is with respect to choice of the hash function. How would you improve the probability that* $\frac{x}{4} \leq \hat{x} \leq 4x$ *to 0.8? Hint: Since we do not know the variance taking average may not help and we need to use some other function of multiple runs.*

Exercise 6.7 *Give an example of a set H of hash functions such that* $h(x)$ *is equally likely to be any element of* $\{0,\ldots,M-1\}$ *(H is 1-universal) but H is not 2-universal.*

Exercise 6.8 *Let p be a prime. A set of hash functions*

$$H = \{h | \{0,1,\ldots,p-1\} \rightarrow \{0,1,\ldots,p-1\}\}$$

is 3-universal if for all x,y,z,u,v,w in $\{0,1,\ldots,p-1\}$*, where x,y,z are distinct we have*

$$Prob\Big(h(x) = u,\ h(y) = v,\ h(z) = w\Big) = \frac{1}{p^3}.$$

(a) *Is the set* $\{h_{ab}(x) = ax + b \bmod p \mid 0 \leq a,b < p\}$ *of hash functions 3-universal?*

(b) *Give a 3-universal set of hash functions.* **Exercise 6.9** *Select a value for*

k and create a set

$$H = \big\{\mathbf{x} | \mathbf{x} = (x_1, x_2, \ldots, x_k), x_i \in \{0, 1, \ldots, k-1\}\big\}$$

where the set of vectors H is pairwise independent and $|H| < k^k$*. We say that a set of vectors is pairwise independent if for any subset of two of the coordinates, all of the* k^2 *possible pairs of values that could appear in those coordinates such as* $(0,0),(0,1),\ldots,(1,0),(1,1),\ldots$ *occur the exact same number of times.*

Exercise 6.10 *Consider a coin that comes down heads with probability p. Prove that the expected number of flips needed to see a heads is* $1/p$.

Exercise 6.11 *Randomly generate a string* $x_1x_2\cdots x_n$ *of* 10^6 *0's and 1's with probability* $1/2$ *of* x_i *being a 1. Count the number of ones in the string and also estimate the number of ones by the coin-flip approximate counting algorithm, in Section 6.2.2. Repeat the process for p=1/4, 1/8, and 1/16. How close is the approximation?*

Counting Frequent Elements The Majority and Frequent Algorithms The Second Moment

Exercise 6.12

1. *Construct an example in which the majority algorithm gives a false positive, i.e., stores a non majority element at the end.*

2. *Construct an example in which the frequent algorithm in fact does as badly as in the theorem, i.e., it under counts some item by n/(k+1).*

Exercise 6.13 *Let p be a prime and $n \geq 2$ be an integer. What representation do you use to do arithmetic in the finite field with p^n elements? How do you do addition? How do you do multiplication?*

Error-Correcting codes, polynomial interpolation and limited-way independence

Exercise 6.14 *Let F be a field. Prove that for any four distinct points a_1, a_2, a_3, and a_4 in F and any four possibly not distinct values b_1, b_2, b_3, and b_4 in F, there is a unique polynomial $f(x) = f_0 + f_1x + f_2x^2 + f_3x^3$ of degree at most three so that $f(a_i) = b_i$, $1 \leq i \leq 4$ with all computations done over F. If you use the Vandermonde matrix you can use the fact that the matrix is nonsingular.*

Sketch of a Large Matrix

Exercise 6.15 *Suppose we want to pick a row of a matrix at random where the probability of picking row i is proportional to the sum of squares of the entries of that row. How would we do this in the streaming model?*

(a) *Do the problem when the matrix is given in column order.*

(b) *Do the problem when the matrix is represented in sparse notation: it is just presented as a list of triples (i,j,a_{ij}), in arbitrary order.*

Matrix Multiplication Using Sampling

Exercise 6.16 *Suppose A and B are two matrices. Prove that* $AB = \sum_{k=1}^{n} A(:,k)B(k,:)$.

Exercise 6.17 *Generate two 100 by 100 matrices A and B with integer values between 1 and 100. Compute the product AB both directly and by sampling. Plot the difference in L_2 norm between the results as a function of the number of samples. In generating the matrices make sure that they are skewed. One method would be the following. First generate two 100 dimensional vectors a and b with integer values between 1 and 100. Next generate the i^{th} row of A with integer values between 1 and a_i and the i^{th} column of B with integer values between 1 and b_i.*

Approximating a Matrix with a Sample of Rows and Columns

Exercise 6.18 *Suppose $a_1, a_2, \ldots, a_m$ are nonnegative reals. Show that the minimum of $\sum_{k=1}^{m} \frac{a_k}{x_k}$ subject to the constraints $x_k \geq 0$ and $\sum_k x_k = 1$ is attained when the x_k are proportional to $\sqrt{a_k}$.*

Sketches of Documents

Exercise 6.19 *Construct two different strings of 0's and 1's having the same set of substrings of length $k = 3$.*

Exercise 6.20 *(Random strings, empirical analysis). Consider random strings of length n composed of the integers 0 through 9, where we represent a string $\mathbf{x}$ by its set $S_k(\mathbf{x})$ of length k-substrings. Perform the following experiment: choose two random strings $\mathbf{x}$ and $\mathbf{y}$ of length $n = 10{,}000$ and compute their resemblance $\frac{|S_k(\mathbf{x}) \cap S_k(\mathbf{y})|}{|S_k(\mathbf{x}) \cup S_k(\mathbf{y})|}$ for $k = 1,2,3\ldots$. What does the graph of resemblance as a function of k look like?*

Exercise 6.21 *(Random strings, theoretical analysis). Consider random strings of length n composed of the integers 0 through 9, where we represent a string $\mathbf{x}$ by its set $S_k(\mathbf{x})$ of length k-substrings. Consider now drawing two random strings $\mathbf{x}$ and $\mathbf{y}$ of length n and computing their resemblance $\frac{|S_k(\mathbf{x}) \cap S_k(\mathbf{y})|}{|S_k(\mathbf{x}) \cup S_k(\mathbf{y})|}$.*

1. *Prove that for $k \leq \frac{1}{2}\log_{10}(n)$, with high probability as n goes to infinity the two strings have resemblance equal to 1.*

2. *Prove that for $k \geq 3\log_{10}(n)$, with high probability as n goes to infinity the two strings have resemblance equal to 0.*

Exercise 6.22 *Discuss how you might go about detecting plagiarism in term papers.*

Exercise 6.23 *Suppose you had one billion web pages and you wished to remove duplicates. How might you do this?*

Exercise 6.24 *Consider the following lyrics:*

> *When you walk through the storm hold your head up high and don't be afraid of the dark. At the end of the storm there's a golden sky and the sweet silver song of the lark.*
> *Walk on, through the wind, walk on through the rain though your dreams be tossed and blown. Walk on, walk on, with hope in your heart and you'll never walk alone, you'll never walk alone.*

How large must k be to uniquely recover the lyric from the set of all subsequences of symbols of length k? Treat the blank as a symbol.

Exercise 6.25 *Blast: Given a long string A, say of length* 10^9 *and a shorter string B, say* 10^5*, how do we find a position in A which is the start of a substring* B^0 *that is close to B? This problem can be solved by dynamic programming in polynomial time, but find a faster algorithm to solve this problem.*
Hint: (Shingling approach) One possible approach would be to fix a small length, say seven, and consider the shingles of A and B of length seven. If a close approximation to B is a substring of A, then a number of shingles of B must be shingles of A. This should allows us to find the approximate location in A of the approximation of B. Some final algorithm should then be able to find the best match.

7 Clustering

7.1 Introduction

Clustering refers to partitioning a set of objects into subsets according to some desired criterion. Often it is an important step in making sense of large amounts of data. Clustering comes up in many contexts. One might want to partition a set of news articles into clusters based on the topics of the articles. Given a set of pictures of people, one might want to group them into clusters based on who is in the image. Or one might want to cluster a set of protein sequences according to the protein function. A related problem is not finding a full partitioning but rather just identifying natural clusters that exist. For example, given a collection of friendship relations among people, one might want to identify any tight-knit groups that exist. In some cases we have a well-defined correct answer, e.g., in clustering photographs of individuals by who is in them, but in other cases the notion of a good clustering may be more subjective.

Before running a clustering algorithm, one first needs to choose an appropriate representation for the data. One common representation is as vectors in R^d. This corresponds to identifying d real-valued features that are then computed for each data object. For example, to represent documents one might use a "bag of words" representation, where each feature corresponds to a word in the English language and the value of the feature is how many times that word appears in the document. Another common representation is as vertices in a graph, with edges weighted by some measure of how similar or dissimilar the two endpoints are. For example, given a set of protein sequences, one might weight edges based on an edit-distance measure that essentially computes the cost of transforming one sequence into the other. This measure is typically symmetric and satisfies the triangle inequality, and so can be thought of as a finite metric. A point worth noting up front is that often the "correct" clustering of a given set of data depends on your goals. For instance, given a set of photographs of individuals, we might want to cluster the images by who is in them, or we might want to cluster them by facial expression. When representing the images as points in space or as nodes in a weighted graph, it is important that the features we use be relevant to the criterion we care about. In any event, the issue of how best to represent data to highlight the relevant information for a given task is generally addressed using knowledge of the specific domain. From our

perspective, the job of the clustering algorithm begins after the data has been represented in some appropriate way.

In this chapter, our goals are to discuss (a) some commonly used clustering algorithms and what one can prove about them, and (b) models and assumptions on data under which we can find a clustering close to the correct clustering.

7.1.1 Preliminaries

We will follow the standard notation of using n to denote the number of data points and k to denote the number of desired clusters. We will primarily focus on the case that k is known up front, but will also discuss algorithms that produce a sequence of solutions, one for each value of k, as well as algorithms that produce a cluster tree that can encode multiple clusterings at each value of k. We will generally use $A = \{\mathbf{a}_1,\ldots,\mathbf{a}_n\}$ to denote the n data points. We also think of A as a matrix with rows $\mathbf{a}_1,\ldots,\mathbf{a}_n$.

7.1.2 Two General Assumptions on the Form of Clusters

Before choosing a clustering algorithm, it is useful to have some general idea of what a good clustering should look like. In general, there are two types of assumptions often made that in turn lead to different classes of clustering algorithms.

Center-based clusters: One assumption commonly made is that clusters are *centerbased*. This means that the clustering can be defined by k centers $\mathbf{c}_1,\ldots,\mathbf{c}_k$, with each data point assigned to whichever center is closest to it. Note that this assumption does not yet tell whether one choice of centers is better than another. For this, one needs an objective, or optimization criterion. Three standard criteria often used are k-center, k-median, and k-means clustering, defined as follows.

k-center clustering: Find a partition $C = \{C_1,\ldots,C_k\}$ of A into k clusters, with corresponding centers $\mathbf{c}_1,\ldots,\mathbf{c}_k$, to minimize the *maximum* distance between any data point and the center of its cluster. That is, we want to minimize

$$\Phi_{kcenter}(\mathcal{C}) = \max_{j=1}^{k} \max_{\mathbf{a}_i \in C_j} d(\mathbf{a}_i, \mathbf{c}_j).$$

k-center clustering makes sense when we believe clusters should be local regions in space. It is also often thought of as the "firehouse location problem" since one can think of it as the problem of locating k fire-stations in a city so as to minimize the maximum distance a fire-truck might need to travel to put out a fire.

k-median clustering: Find a partition $C = \{C_1,\ldots,C_k\}$ of A into k clusters, with corresponding centers $\mathbf{c}_1,\ldots,\mathbf{c}_k$, to minimize the *sum* of distances between data points and the centers of their clusters. That is, we want to minimize

$$k$$

$$\Phi_{kmedian}(C) = \sum_{j=1}^{k} \sum_{\mathbf{a}_i \in C_j} d(\mathbf{a}_i, \mathbf{c}_j).$$

k-median clustering is more noise-tolerant than *k*-center clustering because we are taking a sum rather than a max. A small number of outliers will typically not change the optimal solution by much, unless they are very far away or there are several quite different near-optimal solutions.

k-means clustering: Find a partition $C = \{C_1,...,C_k\}$ of A into k clusters, with corresponding centers $\mathbf{c}_1,...,\mathbf{c}_k$, to minimize the *sum of squares* of distances between data points and the centers of their clusters. That is, we want to minimize

$$\Phi_{kmeans}(C) = \sum_{j=1}^{k} \sum_{\mathbf{a}_i \in C_j} d_2(\mathbf{a}_i, \mathbf{c}_j).$$

k-means clustering puts more weight on outliers than *k*-median clustering, because we are squaring the distances, which magnifies large values. This puts it somewhat in between *k*-median and *k*-center clustering in that regard. Using distance squared has some mathematical advantages over using pure distances when data are points in R^d. For example, Corollary 7.2 that asserts that with the distance squared criterion, the optimal center for a given group of data points is its centroid.

The *k*-means criterion is more often used when data consists of points in R^d, whereas *k*-median is more commonly used when we have a finite metric, that is, data are nodes in a graph with distances on edges.

When data are points in R^d, there are in general two variations of the clustering problem for each of the criteria. We could require that each cluster center be a data point or allow a cluster center to be any point in space. If we require each center to be a data point, the optimal clustering of n data points into k clusters can be solved in time $\binom{n}{k}$ times a polynomial in the length of the data. First, exhaustively enumerate all sets of k data points as the possible sets of k cluster centers, then associate each point to its nearest center and select the best clustering. No such naive enumeration procedure is available when cluster centers can be any point in space. But, for the *k*-means problem, Corollary 7.2 shows that once we have identified the data points that belong to a cluster, the best choice of cluster center is the centroid of that cluster, which might not be a data point.

When k is part of the input or may be a function of n, the above optimization problems are all NP-hard.[33] So, guarantees on algorithms will typically involve either some form of approximation or some additional assumptions, or both.

High-density clusters: If we do not believe our desired clusters will be center-based, an alternative assumption often made is that clusters consist of high-density regions surrounded by low-density "moats" between them. For example, in the clustering of Figure 7.1 we have one natural cluster A that looks center-based but the other cluster B consists of a ring around cluster A. As seen in the figure, this assumption does not require clusters to correspond to convex regions and it can allow them to be long and stringy. We describe a non-center-based clustering method in Section 7.7. In Section 7.9 we prove the effectiveness of an algorithm which finds a "moat", cuts up data "inside" the moat and 'outside" into two pieces and recursively applies the same procedure to each piece.

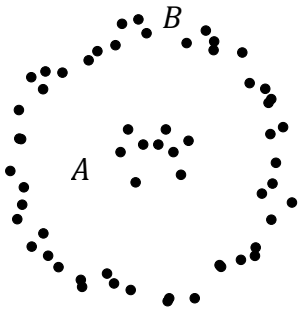


Figure 7.1: Example where the natural clustering is not center-based.

7.1.3 Spectral Clustering

An important part of a clustering toolkit when data lies in R^d is Singular Value Decomposition. Spectral Clustering refers to the following algorithm: Find the space V spanned by the top k right singular vectors of the matrix A whose rows are the data points. Project data points to V and cluster in the projection.

An obvious reason to do this is dimension reduction, clustering in the d dimensional space where data lies is reduced to clustering in a k dimensional space (usually, $k << d$). A more important point is that under certain assumptions one can prove that spectral clustering gives a clustering close to the true clustering. We already saw this in the case when data is from a mixture of spherical Gaussians, Section 3.9.3. The assumption used is "the means separated by a constant number of Standard Deviations". In Section 7.5, we will see that in a much more general setting, which includes common stochastic models, the same assumption, in spirit, yields similar conclusions. Section 7.4, has another setting with a similar result.

[33] If k is a constant, then as noted above, the version where the centers must be data points can be solved in polynomial time.

7.2 k-Means Clustering

We assume in this section that data points lie in R^d and focus on the k-means criterion.

7.2.1 A Maximum-Likelihood Motivation

We now consider a maximum-likelihood motivation for using the k-means criterion. Suppose that the data was generated according to an equal weight mixture of k spherical well-separated Gaussian densities centered at $\mu_1,\mu_2,\ldots,\mu_k$, each with variance one in every direction. Then the density of the mixture is

$$\text{Prob}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}} \frac{1}{k} \sum_{i=1}^{k} e^{-|\mathbf{x}-\boldsymbol{\mu}_i|^2}.$$

Denote by $\mu(\mathbf{x})$ the center nearest to $\mathbf{x}$. Since the exponential function falls off fast, assuming x is noticeably closer to its nearest center than to any other center, we can approximate $\sum_{i=1}^{k} e^{-|\mathbf{x}-\boldsymbol{\mu}_i|^2}$ by $e^{-|\mathbf{x}-\mu(\mathbf{x})|_2}$ since the sum is dominated by its largest term. Thus

$$\text{Prob}(\mathbf{x}) \approx \frac{1}{(2\pi)^{d/2}k} e^{-|\mathbf{x}-\boldsymbol{\mu}(\mathbf{x})|^2}.$$

The likelihood of drawing the sample of points $\mathbf{x_1},\mathbf{x_2},\ldots,\mathbf{x_n}$ from the mixture, if the centers were $\mu_1,\mu_2,\ldots,\mu_k$, is approximately

$$\frac{1}{k^n} \frac{1}{(2\pi)^{nd/2}} \prod_{i=1}^{n} e^{-|\mathbf{x}^{(i)}-\boldsymbol{\mu}(\mathbf{x}^{(i)})|^2} = ce^{-\sum_{i=1}^{n} |\mathbf{x}^{(i)}-\boldsymbol{\mu}(\mathbf{x}^{(i)})|^2}.$$

Minimizing the sum of squared distances to cluster centers finds the maximum likelihood $\mu_1,\mu_2,\ldots,\mu_k$. This motivates using the sum of distance squared to the cluster centers.

7.2.2 Structural Properties of the k-Means Objective

Suppose we have already determined the clustering or the partitioning into $C_1,C_2,\ldots,C_k$. What are the best centers for the clusters? The following lemma shows that the answer is the centroids, the coordinate means, of the clusters.

Lemma 7.1 *Let* $\{\mathbf{a_1},\mathbf{a_2},\ldots,\mathbf{a_n}\}$ *be a set of points. The sum of the squared distances of the* $\mathbf{a_i}$ *to any point* $\mathbf{x}$ *equals the sum of the squared distances to the centroid of the* $\mathbf{a_i}$ *plus n times the squared distance from* $\mathbf{x}$ *to the centroid. That is,*

$$\sum_i |\mathbf{a_i} - \mathbf{x}|^2 = \sum_i |\mathbf{a_i} - \mathbf{c}|^2 + n|\mathbf{c} - \mathbf{x}|^2$$

where $\mathbf{c} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{a_i}$ *is the centroid of the set of points.*

Proof:

$$\sum_i |\mathbf{a_i} - \mathbf{x}| = \sum_i |\mathbf{a_i} - \mathbf{c} + \mathbf{c} - \mathbf{x}|^2$$

$$= \sum_i |\mathbf{a_i} - \mathbf{c}|^2 + 2(\mathbf{c} - \mathbf{x}) \cdot \sum_i (\mathbf{a_i} - \mathbf{c}) + n|\mathbf{c} - \mathbf{x}|^2$$

Since $\mathbf{c}$ is the centroid, $\sum_i (\mathbf{a_i} - \mathbf{c}) = 0$. Thus, $\sum_i |\mathbf{a_i} - \mathbf{x}|^2 = \sum_i |\mathbf{a_i} - \mathbf{c}|^2 + n|\mathbf{c} - \mathbf{x}|^2$ ∎

A corollary of Lemma 7.1 is that the centroid minimizes the sum of squared distances since the first term, $\sum_i |\mathbf{a_i} - \mathbf{c}|^2$, is a constant independent of $\mathbf{x}$ and setting $\mathbf{x} = \mathbf{c}$ sets the second term, $n\|\mathbf{c} - \mathbf{x}\|^2$, to zero.

Corollary 7.2 *Let* $\{\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n}\}$ *be a set of points. The sum of squared distances of the* $\mathbf{a_i}$ *to a point* $\mathbf{x}$ *is minimized when* $\mathbf{x}$ *is the centroid, namely* $\mathbf{x} = \frac{1}{n} \sum_i \mathbf{a_i}$.

7.2.3 Lloyd's Algorithm

Corollary 7.2 suggests the following natural strategy for *k*-means clustering, known as Lloyd's algorithm. Lloyd's algorithm does not necessarily find a globally optimal solution but will find a locally-optimal one. An important but unspecified step in the algorithm is its initialization: how the starting *k* centers are chosen. We discuss this after discussing the main algorithm.

Lloyd's algorithm:

Start with *k* centers.

Cluster each point with the center nearest to it.

Find the centroid of each cluster and replace the set of old centers with the centroids.

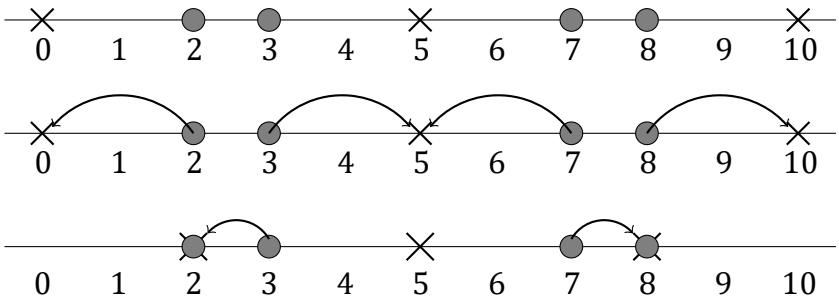
Repeat the above two steps until the centers converge according to some criterion, such as the *k*-means score no longer improving.

This algorithm always converges to a local minimum of the objective. To show convergence, we argue that the sum of the squares of the distances of each point to its cluster center always improves. Each iteration consists of two steps. First, consider the step that finds the centroid of each cluster and replaces the old centers with the new centers. By Corollary 7.2, this step improves the sum of internal cluster distances squared. The second step reclusters by assigning each point to its nearest cluster center, which also improves the internal cluster distances.

A problem that arises with some implementations of the *k*-means clustering algorithm is that one or more of the clusters becomes empty and there is no center from which to

measure distance. A simple case where this occurs is illustrated in the following example. You might think how you would modify the code to resolve this issue.

Example: Consider running the *k*-means clustering algorithm to find three clusters on the following 1-dimension data set: {2,3,7,8} starting with centers {0,5,10}.



The center at 5 ends up with no items and there are only two clusters instead of the desired three. ■

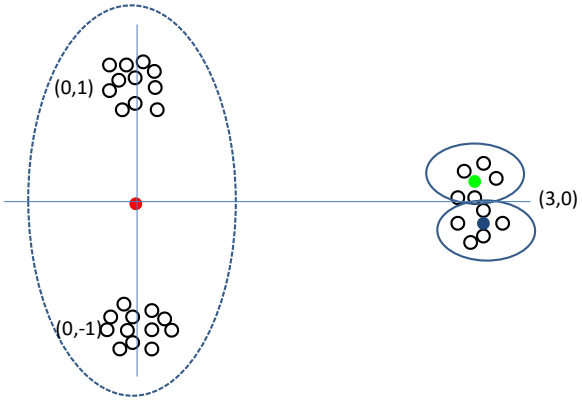


Figure 7.2: A locally-optimal but globally-suboptimal *k*-means clustering.

As noted above, Lloyd's algorithm only finds a local optimum to the *k*-means objective that might not be globally optimal. Consider, for example, Figure 7.2. Here data lies in three dense clusters in R^2: one centered at (0,1), one centered at (0,−1) and one centered at (3,0). If we initialize with one center at (0,1) and two centers near (3,0), then the center at (0,1) will move to near (0,0) and capture the points near (0,1) and (0,−1), whereas the centers near (3,0) will just stay there, splitting that cluster.

Because the initial centers can substantially influence the quality of the result, there has been significant work on initialization strategies for Lloyd's algorithm. One popular strategy is called "farthest traversal". Here, we begin by choosing one data point as initial center $\mathbf{c}_1$ (say, randomly), then pick the farthest data point from $\mathbf{c}_1$ to use as $\mathbf{c}_2$, then pick the farthest data point from $\{\mathbf{c}_1,\mathbf{c}_2\}$ to use as $\mathbf{c}_3$, and so on. These are then used as the initial centers. Notice that this will produce the correct solution in the example in Figure 7.2.

Farthest traversal can unfortunately get fooled by a small number of outliers. To address this, a smoother, probabilistic variation known as k-means++ instead weights data points based on their distance squared from the previously chosen centers. Then it selects the next center probabilistically according to these weights. This approach has the nice property that a small number of outliers will not overly influence the algorithm so long as they are not too far away, in which case perhaps they should be their own clusters anyway.

Another approach is to run some other approximation algorithm for the k-means problem, and then use its output as the starting point for Lloyd's algorithm. Note that applying Lloyd's algorithm to the output of any other algorithm can only improve its score. An alternative SVD-based method for initialization is described and analyzed in Section 7.5.

7.2.4 Ward's Algorithm

Another popular heuristic for k-means clustering is Ward's algorithm. Ward's algorithm begins with each data point in its own cluster, and then repeatedly merges pairs of clusters until only k clusters remain. Specifically, Ward's algorithm merges the two clusters that minimize the immediate increase in k-means cost. That is, for a cluster C, define $cost(C) = \sum_{\mathbf{a}_i \in C} d^2(\mathbf{a}_i, \mathbf{c})$, where $\mathbf{c}$ is the centroid of C. Then Ward's algorithm merges the pair (C, C^0) minimizing $cost(C \cup C^0) - cost(C) - cost(C^0)$. Thus, Ward's algorithm can be viewed as a greedy k-means algorithm.

7.2.5 k-Means Clustering on the Line

One case where the optimal k-means clustering can be found in polynomial time is when points lie in R^1, i.e., on the line. This can be done using dynamic programming, as follows.

First, assume without loss of generality that the data points $a_1,\ldots,a_n$ have been sorted, so $a_1 \le a_2 \le \ldots \le a_n$. Now, suppose that for some $i \ge 1$ we have already computed the optimal k^0-means clustering for points $a_1,\ldots,a_i$ for all $k^0 \le k$; note that this is trivial to do for the base case of $i = 1$. Our goal is to extend this solution to points $a_1,\ldots,a_{i+1}$. To do so, observe that each cluster will contain a consecutive sequence of data points. So, given k^0, for each $j \le i + 1$, compute the cost of using a single center for points $a_j,\ldots,a_{i+1}$, which is the sum of distances of each of these points to their mean value. Then add to that the cost of the optimal $k^0 - 1$ clustering of points $a_1,\ldots,a_{j-1}$ which we computed earlier. Store the minimum of these sums, over choices of j, as our optimal k^0-means clustering of points $a_1,\ldots,a_{i+1}$. This has running time of $O(kn)$ for a given value of i. So overall our running time is $O(kn^2)$.

7.3 k-Center Clustering

In this section, instead of using the k-means clustering criterion, we use the k-center criterion. Recall that the k-center criterion partitions the points into k clusters so as to

minimize the maximum distance of any point to its cluster center. Call the maximum distance of any point to its cluster center the *radius* of the clustering. There is a k-clustering of radius r if and only if there are k spheres, each of radius r, which together cover all the points. Below, we give a simple algorithm to find k spheres covering a set of points. The following lemma shows that this algorithm only needs to use a radius that is at most twice that of the optimal k-center solution. Note that this algorithm is equivalent to the farthest traversal strategy for initializing Lloyd's algorithm.

The Farthest Traversal k-clustering Algorithm

> Pick any data point to be the first cluster center. At time t, for $t = 2,3,\ldots,k$, pick the farthest data point from any existing cluster center; make it the t^{th} cluster center.

Theorem 7.3 *If there is a k-clustering of radius $\frac{r}{2}$, then the above algorithm finds a k-clustering with radius at most r.*

Proof: Suppose for contradiction that there is some data point $\mathbf{x}$ that is distance greater than r from all centers chosen. This means that each new center chosen was distance greater than r from all previous centers, because we could always have chosen $\mathbf{x}$. This implies that we have $k+1$ data points, namely the centers chosen plus $\mathbf{x}$, that are pairwise more than distance r apart. Clearly, no two such points can belong to the same cluster in any k-clustering of radius $\frac{r}{2}$, contradicting the hypothesis. ■

7.4 Finding Low-Error Clusterings

In the previous sections we saw algorithms for finding a local optimum to the k-means clustering objective, for finding a global optimum to the k-means objective on the line, and for finding a factor 2 approximation to the k-center objective. But what about finding a clustering that is close to the correct answer, such as the true clustering of proteins by function or a correct clustering of news articles by topic? For this we need some assumption about the data and what the correct answer looks like. The next few sections consider algorithms based on different such assumptions.

7.5 Spectral Clustering

Let A be a $n \times d$ data matrix with each row a data point and suppose we want to partition the data points into k clusters. *Spectral Clustering* refers to a class of clustering algorithms which share the following outline:

- Find the space V spanned by the top k (right) singular vectors of A.
- Project data points into V.
- Cluster the projected points.

7.5.1 Why Project?

The reader may want to read Section 3.9.3, which shows the efficacy of spectral clustering for data stochastically generated from a mixture of spherical Gaussians. Here, we look at general data which may not have a stochastic generation model.

We will later describe the last step in more detail. First, lets understand the central advantage of doing the projection to V. It is simply that for any reasonable (unknown) clustering of data points, the projection brings data points closer to their cluster centers. This statement sounds mysterious and likely false, since the assertion is for ANY reasonable unknown clustering. We quantify it in Theorem 7.4. First some notation: We represent a k-clustering by a $n \times d$ matrix C (same dimensions as A), where row i of C is

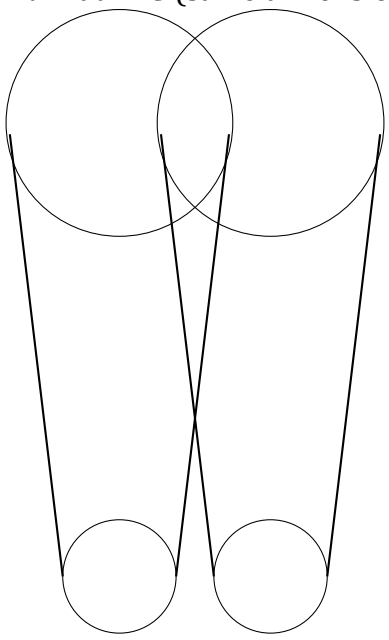


Figure 7.3: Clusters in the full space and their projections

the center of the cluster to which data point i belongs. So, there are only k distinct rows of C and each other row is a copy of one of these rows. The k-means objective function, namely, the sum of squares of the distances of data points to their cluster centers is

$$\sum_{i=1}^{n} |\mathbf{a_i} - \mathbf{c_i}|^2 = ||A - C||_F^2 .$$

The projection reduces the sum of distance squares to cluster centers from $||A - C||_F^2$ to at most $8^k||A - C||_2^2$ in the projection. Recall that $||A - C||_2$ is the spectral norm, which is the top singular value of $A - C$. Now, $||A - C||_F^2 = \sum_t \sigma_t^2(A)$ and often,

$||A - C||_F >> k||A - C||_2$ and so the projection substantially reduces the sum of squared distances to cluster centers.

We will see later that in many clustering problems, including models like mixtures of Gaussians and Stochastic Block Models of communities, there is a desired clustering C where the regions overlap in the whole space, but are separated in the projection. Figure 7.3 is a schematic illustration. Now we state the theorem and give its surprisingly simple proof.

Theorem 7.4 *Let A be an n × d matrix with A_k the projection of the rows of A to the subspace of the first k right singular vectors of A. For any matrix C of rank less than or equal to k*

$$||A_k - C||_F^2 \leq 8k||A - C||_2^2.$$

A_k is a matrix that is close to every C, in the sense $||A_k - C||_F^2 \leq 8k||A - C||_2^2$. While this seems contradictory, another way to state this is that for C far away from A_k in Frobenius norm, $||A - C||_2$ will also be high.

Proof: Since the rank of $(A_k - C)$ is less than or equal to $2k$,

$$||A_k - C||_F^2 \leq 2k||A_k - C||_2^2 \quad \text{and}$$

$$||A_k - C||_2 \leq ||A_k - A||_2 + ||A - C||_2 \leq 2||A - C||_2.$$

The last inequality follows since A_k is the best rank k approximation in spectral norm (Theorem 3.9) and C has rank at most k. The theorem follows. ■

Suppose now in the clustering C we would like to find, the cluster centers that are pairwise at least $\Omega(\sqrt{k}||A - C||_2)$ apart. This holds for many clustering problems including data generated by stochastic models. Then, it will be easy to see that in the projection, most data points are a constant factor farther from centers of other clusters than their own cluster center and this makes it very easy for the following algorithm to find the clustering C modulo a small fraction of errors.

7.5.2 The Algorithm

Denote $||A-C||_2/\sqrt{n}$ by $\sigma(C)$. In the next section, we give an interpretation of $||A-C||_2$ indicating that $\sigma(C)$ is akin to the standard deviation of clustering C and hence the notation $\sigma(C)$. We assume for now that $\sigma(C)$ is known to us for the desired clustering C.
This assumption can be removed by essentially doing a binary search. **Spectral Clustering - The Algorithm**

1. Find the top k right singular vectors of data matrix A and project rows of A to the space spanned by them to get A_k.(cf. Section 3.5).

2. Select a random row from A_k and form a cluster with all rows of A_k at distance less than $6k\sigma(C)/\varepsilon$ from it.

3. Repeat Step 2 k times.

Theorem 7.5 *If in a k-clustering C, every pair of centers is separated by at least* $15k\sigma(C)/\varepsilon$ *and every cluster has at least* εn *points in it, then with probability at least* $1 - \varepsilon$*, Spectral Clustering finds a clustering* C^0 *that differs from C on at most* $\varepsilon^2 n$ *points.*

Proof: Let $\mathbf{v_i}$ denote row i of A_k. We first show that for most data points, the projection of data point is within distance $3k\sigma(C)/\varepsilon$ of its cluster center. I.e., we show that $|M|$ is small, where,

$$M = \{i : |\mathbf{v_i} - \mathbf{c_i}| \geq 3k\sigma(C)/\varepsilon\}.$$

Now, $||A_k - C||_F^2 = \sum_i |\mathbf{v_i} - \mathbf{c_i}|^2 \geq \sum_{i \in M} |\mathbf{v_i} - \mathbf{c_i}|^2 \geq |M| \frac{9k^2\sigma^2(C)}{\varepsilon^2}$. So, using Theorem 7.4, we get:

$$|M| \frac{9k^2\sigma^2(C)}{\varepsilon^2} \leq ||A_k - C||_F^2 \leq 8kn\sigma^2(C) \quad \Longrightarrow \quad |M| \leq \frac{8\varepsilon^2 n}{9k}. \tag{7.1}$$

Call a data point i "good" if $i \notin M$. For any two good data points i and j belonging to the same cluster, since, their projections are within $3k\sigma(C)/\varepsilon$ of the center of the cluster, projections of the two data points are within $6k\sigma(C)/\varepsilon$ of each other. On the other hand, if two good data points i and k are in different clusters, since, the centers of the two clusters are at least $15k\sigma(C)/\varepsilon$ apart, their projections must be greater than $15k\sigma(C)/\varepsilon - 6k\sigma(C)/\varepsilon = 9k\sigma(C)/\varepsilon$ apart. So, if we picked a good data point (say point i) in Step 2, the set of good points we put in its cluster is exactly the set of good points in the same cluster as i. Thus, if in each of the k executions of Step 2, we picked a good point, all good points are correctly clustered and since $|M| \leq \varepsilon^2 n$, the theorem would hold.

To complete the proof, we must argue that the probability of any pick in Step 2 being bad is small. The probability that the first pick in Step 2 is bad is at most $|M|/n \leq \varepsilon^2/k$. For each subsequent execution of Step 2, all the good points in at least one cluster are remaining candidates. So there are at least $(\varepsilon - \varepsilon^2)n$ good points left and so the probability that we pick a bad point is at most $|M|/(\varepsilon - \varepsilon^2)n$ which is at most ε/k. The union bound over the k executions yields the desired result.

■

7.5.3 Means Separated by $\Omega(1)$ Standard Deviations

For probability distribution on the real line, the mnemonic "means separated by six standard deviations" suffices to distinguish different distributions. Spectral Clustering enables us to do the same thing in higher dimensions provided $k \in O(1)$ and six is replaced by some constant. First we define standard deviation for general not necessarily stochastically generated data: it is just the maximum over all unit vectors $\mathbf{v}$ of the square root of the mean squared distance of data points from their cluster centers in the direction $\mathbf{v}$, namely, the standard deviation $\sigma(C)$ of clustering C is defined as:

$$\sigma(C)^2 = \frac{1}{n} \text{Max}_{\mathbf{v}:|\mathbf{v}|=1} \sum_{i=1}^{n} [(\mathbf{a_i} - \mathbf{c_i}) \cdot \mathbf{v}]^2 = \frac{1}{n} \text{Max}_{\mathbf{v}:|\mathbf{v}|=1} |(A - C)\mathbf{v}|^2 = \frac{1}{n} ||A - C||_2^2.$$

This coincides with the definition of $\sigma(C)$ we made earlier. Assuming $k \in O(1)$, it is easy to see that the Theorem 7.5 can be restated as

> If cluster centers in C are separated by $\Omega(\sigma(C))$, then the spectral clustering algorithm finds C^0 which differs from C only in a small fraction of data points.

It can be seen that the "means separated by $\Omega(1)$ standard deviations" condition holds for many stochastic models. We illustrate with two examples here. First, suppose we have a mixture of $k \in O(1)$ spherical Gaussians, each with standard deviation one. The data is generated according to this mixture. If the means of the Gaussians are $\Omega(1)$ apart, then the condition - means separated by $\Omega(1)$ standard deviations- is satisfied and so if we project to the SVD subspace and cluster, we will get (nearly) the correct clustering. This was already discussed in detail in Chapter **??**.

We discuss a second example. *Stochastic Block Models* are models of communities. Suppose there are k communities $C_1, C_2, \ldots, C_k$ among a population of n people. Suppose the probability of two people in the same community knowing each other is p and if they are in different communities, the probability is q, where, $q < p$.[34] We assume the events that person i knows person j are *independent* across all i and j.

Specifically, we are given an $n \times n$ data matrix A, where $a_{ij} = 1$ if and only if i and j know each other. We assume the a_{ij} are independent random variables, and use $\mathbf{a_i}$ to denote the i^{th} row of A. It is useful to think of A as the adjacency matrix of a graph, such as the friendship network in Facebook. We will also think of the rows $\mathbf{a_i}$ as data points. The clustering problem is to classify the data points into the communities they belong to. In practice, the graph is fairly sparse, i.e., p and q are small, namely, $O(1/n)$ or $O(\ln n/n)$. Consider the simple case of two communities with $n/2$ people in each and with

$$p = \frac{\alpha}{n} \qquad q = \frac{\beta}{n} \qquad \text{where } \alpha, \beta \in O(\ln n).$$

Let $\mathbf{u}$ and $\mathbf{v}$ be the centroids of the data points in community one and community two respectively; so $u_i \approx p$ for $i \in C_1$ and $u_j \approx q$ for $j \in C_2$ and $v_i \approx q$ for $i \in C_1$ and $v_j \approx p$ for $j \in C_2$. We have

$$|\mathbf{u} - \mathbf{v}|^2 = \sum_{j=1}^{n} (u_j - v_j)^2 \approx \frac{(\alpha - \beta)^2}{n^2} n = \frac{(\alpha - \beta)^2}{n}.$$

$$\text{Inter-centroid distance} \approx \frac{\alpha - \beta}{\sqrt{n}}. \tag{7.2}$$

[34] More generally, for each pair of communities a and b, there could be a probability p_{ab} that a person from community a knows a person from community b. But for the discussion here, we take $p_{aa} = p$ for all a and $p_{ab} = q$, for all $a \neq b$.

We need to upper bound $||A - C||_2$. This is non-trivial since we have to prove a uniform upper bound on $|(A - C)\mathbf{v}|$ for all unit vectors $\mathbf{v}$. Fortunately, the subject to Random Matrix Theory (RMT) already does this for us. RMT tells that

$$||A - C||_2 \leq O^*(\sqrt{np}) = O^*(\sqrt{\alpha}),$$

O^* hides logarithmic factors. So as long as $\alpha - \beta \in \Omega^*(\sqrt{\alpha})$, we have the where, the O means separated by $\Omega(1)$ standard deviations and spectral clustering works.

One important observation is that in these examples as well as many others, the $k$means objective function in the whole space is too high and so the projection is essential before we can cluster.

7.5.4 Laplacians

An important special case of spectral clustering is when $k = 2$ and a spectral algorithm is applied to the Laplacian matrix L of a graph, which is defined as

$$L = D - A$$

where A is the adjacency matrix and D is a diagonal matrix of degrees. Since A has a negative sign, we look at the lowest two singular values and corresponding vectors rather than the highest,

L is a symmetric matrix and is easily seen to be posiitve semi-definite: for any vector $\mathbf{x}$, we have

$$\mathbf{x}^T L\mathbf{x} = \sum_i d_{ii}x_i^2 - \sum_{(i,j)\in E} x_i x_j = \frac{1}{2}\sum_{(i,j)\in E}(x_i - x_j)^2.$$

Also since all row sums of L (and L is symmetric) are zero, its lowest eignvalue is 0 with the eigenvector $\mathbf{1}$ of all 1's. This is also the lowest singular vector of L. The projection of all data points (rows) to this vector is just the origin and so gives no information. If we take the second lowest singular vector and project to it which is essentially projecting to the space of the bottom two singular vectors, we get the very simple problem of n real numbers which we need to cluster into two clusters.

7.5.5 Local spectral clustering

So far our emphasis has been on partitioning the data into disjoint clusters. However, the structure of many data sets consists of over lapping communities. In this case using k-means with spectral clustering, the overlap of two communities shows up as a community. This is illustrated in Figure 7.4.

An alternative to using *k*-means with spectral clustering is to find the minimum 1norm vector in the space spanned by the top singular vectors. Let *A* be the matrix whose columns are the singular vectors. To find a vector *y* in the space spanned by the columns of *A* solve the linear system $Ax = y$. This is a slightly different looking problem then $Ax = c$ where *c* is a constant vector. To convert $Ax = y$ to the more usual form write it as $[A, -I]\begin{pmatrix} x \\ y \end{pmatrix} = 0$. However, if we want to minimize $||y||_1$ the solution is $x = y = 0$.
Thus we add the row 1,1,...1,0,0,...0 to $[A,-I]$ and a 1 to the top of the vector $[x,y]$ to force the coordinates of *x* to add up to one. Minimizing $||y||_1$ does not appear to be a linear program but we can write $y = y_a - y_b$ and require $y_a \geq 0$ and $y_b \geq 0$. Now finding the minimum one norm vector in the span of the columns of *A* is the linear program

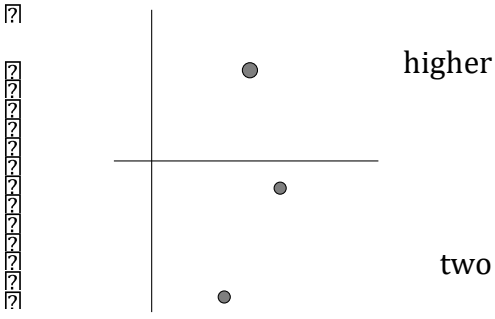
$$\min\left(\sum_i y_{ai} + \sum_i y_{bi}\right) \quad \text{subject to} \quad (A,. \quad -I, I)\begin{pmatrix} x \\ y_a \\ y_b \end{pmatrix} = 0 \quad y_a \geq 0 \quad y_b \geq 0$$

Local communities

In large social networks with a billion vertices, global clustering is likely to result in communities of several hundred million vertices. What you may actually want is a local community containing several individuals with only 50 vertices. To do this if one starts a random walk at a vertex *v* and computes the frequency of visiting vertices, it will converge to the first singular vector. However, the distribution after a small number of steps will be primarily in the small community containing *v* and will be proportional to the first singular vector distribution restricted to the vertices in the small community containing *v*, only will be higher by some constant value. If one wants to determine the local communities containing vertices v_1,v_2, and v_3, start with three probability distributions, one with probability one at v_1, one with probability one at v_2, and one with probability one at v_3 and find early approximation to the first three singular vectors. Then find the minimum 1-norm vector in the space spanned by the early approximations.

Hidden structure

In the previous section we discussed overlapping communities. Another issue is hidden structure. Suppose the vertices of a social network could be partitioned into a number of strongly connected communities. By strongly connected we mean the probability of an edge between two vertices in a community is much than the probability of an edge between two vertices in different communities. Suppose the vertices of the graph could be partitioned in another way which was incoherent[35] with the first partitioning and the probability of an edge between



[35] incoherent, give definition

vertices in one of these communities is higher than an edge between two vertices in different communities. If the probability of an edge between two vertices in a community of this second partitioning is less than that in the first, then a clustering algorithm is likely to find the first partitioning rather than the second. However, the second partitioning, which we refer to as hidden structure, may be the structure that we want to find. The way to do this is to use your favorite clustering algorithm to produce the dominant structure and then stochastically weaken the dominant structure by removing some community edges in the graph. Now if you apply the clustering algorithm to the modified graph, it can find the hidden community structure. Having done this, go back to the original graph, weaken the hidden structure and reapply the clustering algorithm. It will now find a better approximation to the dominant structure. This technology can be used to find a number of hidden levels in several types of social networks. □ 1 1 1 1 1 1 0 0 0 □ □ 0.33 0.31

□ 1 1 1 1 1 1 0 0 0 □ □ 0.33 0.31 □ 1

1 1 1 1 1 0 0 0 □□ □□ 0.33 0.31

□ □ 1 1 1 1 1 1 0 0 0 □□ □□ 0.33 0.31

□ □ 1 1 1 1 1 1 1 1 1 □□ □□ 0.44 −0.09

□ □ 1 1 1 1 1 1 1 1 1 □□ □□ 0.44 −0.09

□

□ □ −0.49

□ 0 0 0 0 1 1 0 0 0 □ □ 0.24

□ □ −0.49

□ 0 0 0 0 1 1 0 0 0 □ □ 0.24

0 0 0 0 1 1 0 0 0 0.24 −0.49

(a) (b) (c);

Figure 7.4: (a) illustrates the adjacency matrix of a graph with a six vertex clique that overlaps a five vertex clique in two vertices. (b) illustrates the matrix where columns consist of the top two singular vectors, and (c) illustrates the mapping of rows in the singular vector matrix to three points in two dimensional space. Instead of two cliques we get the non overlapping portion of each of the two clique plus their intersection as communities instead of the two cliques as communities.

Block model

One technique for generating graphs with communities is to use the block model where the vertices are partitioned into blocks and each block is a GNP graph generated with some edge probability. The edges in off diagonal blocks are generated with a lower probability. One can also generate graphs with hidden structure. For example, the vertices

in an n vertex graph might be partitioned into two communities, the first community having vertices 1 to $n/2$ and the second community having vertices $n/2 + 1$ to n. The dominant structure is generated with probability p_1 for edges within communities and probability q_1 for edges between communities. The the vertices are randomly permuted and the hidden structure is generated using the first $n/2$ vertices in the permuted order for one community and the remaining vertices for the second community with probabilities p_2 and q_2 which ar lower than p_1 and q_1..

An interesting question is how to determine the quality of a community found. Many researchers use an existing standard of what the communities are. However, if you want to using clustering techniques to find communities there probably is no external standard or you would just use that instead of clustering. A way to determine if you have found a real community structure is to ask if the graph is more likely generated by a model of the structure found than by a completely random model. Suppose you found a partition of two communities each with $n/2$ vertices. Using the number of edges in each community and the number of inter community edges ask what is the probability of the graph being generated by a bloc model where p and q are the probabilities determined by the edge density within communities and the edge density between communities. One can compare this probability with the probability that the graph was generated by a GNP model with probability $(p + q)/2$.

7.6 Approximation Stability

7.6.1 The Conceptual Idea

We now consider another condition that will allow us to produce accurate clusters from data. To think about this condition, imagine that we are given a few thousand news articles that we want to cluster by topic. These articles could be represented as points in a high-dimensional space (e.g., axes could correspond to different meaningful words, with coordinate i indicating the frequency of that word in a given article). Or, alternatively, it could be that we have developed some text-processing program that given two articles x and y computes some measure of distance $d(x,y)$ between them. We assume there exists some correct clustering C_T of our news articles into k topics; of course, we do not know what C_T is, that is what we want our algorithm to find.

If we are clustering with an algorithm that aims to minimize the k-means score of its solution, then implicitly this means we believe that the clustering $C_{kmeans^{OPT}}$ of minimum $k$means score is either equal to, or very similar to, the clustering C_T. Unfortunately, finding the clustering of minimum k-means score is NP-hard. So, let us broaden our belief a bit and assume that any clustering C whose k-means score is within 10% of the minimum is also very similar to C_T. This should give us a little bit more slack. Unfortunately, finding a clustering of score within 10% of the minimum is also an NP-hard problem. Nonetheless, *we will be able to use this assumption to efficiently find a clustering that is close to* C_T. The trick is that NP-hardness is a worst-case notion, whereas in contrast, this assumption implies structure on our data.In particular, it implies that all clusterings that have score

within 10% of the minimum have to be similar to each other. We will then be able to utilize this structure in a natural "ball-growing" clustering algorithm.

7.6.2 Making this Formal

To make this discussion formal, we first specify what we mean when we say that two different ways of clustering some data are "similar" to each other. Let C = $\{C_1,\ldots,C_k\}$ and $\mathcal{C}' = \{C'_1, \ldots, C'_k\}$ be two different *k*-clusterings of some dataset *A.* For example, C could be the clustering that our algorithm produces, and C^0 could be the clustering C_T. Let us define the distance between these two clusterings to be the fraction of points that would have to be re-clustered in C to make C match C^0, where by "match" we mean that there should be a bijection between the clusters of C and the clusters of C^0. We can write this distance mathematically as:

$$dist(\mathcal{C}, \mathcal{C}') = \min_{\sigma} \frac{1}{n} \sum_{i=1}^{k} |C_i \setminus C'_{\sigma(i)}|,$$

where the minimum is over all permutations σ of $\{1,\ldots,k\}$.

For $c > 1$ and $\epsilon > 0$ we say that a data set satisfies (c, ϵ)-*approximation-stability* with respect to a given objective (such as *k*-means or *k*-median) if every clustering C whose cost is within a factor *c* of the minimum-cost clustering for that objective satisfies $dist(\mathcal{C}, \mathcal{C}_T) < \epsilon$. That is, it is sufficient to be within a factor *c* of optimal to the our objective in order for the fraction of points clustered incorrectly to be less than . We will specifically focus in this discussion on the *k-median* objective rather than the *k*-means objective, since it is a bit easier to work with.

What we will now show is that under this condition, even though it may be NP-hard in general to find a clustering that is within a factor *c* of optimal, we can nonetheless efficiently find a clustering C^0 such that $dist(\mathcal{C}', \mathcal{C}_T) \leq \epsilon$, so long as all clusters in C_T are reasonably large. To simplify notation, let C^* denote the clustering of minimum *k*-median cost, and to keep the discussion simpler, let us also assume that $C_T = C^*$; that is, the target clustering is also the clustering with the minimum *k*-median score.

7.6.3 Algorithm and Analysis

Before presenting an algorithm, we begin with a helpful lemma that will guide our design. For a given data point $\mathbf{a}_i$, define its *weight* $w(\mathbf{a}_i)$ to be its distance to the center of its cluster in C^*. Notice that the *k*-median cost of $\mathcal{C}^*$ is $OPT = \sum_{i=1}^{n} w(\mathbf{a}_i)$. Define $w_{avg} = OPT/n$ to be the average weight of the points in *A*. Finally, define $w_2(\mathbf{a}_i)$ to be the distance of $\mathbf{a}_i$ to its second-closest center in C^*.

Lemma 7.6 *Assume dataset A satisfies* (c, ϵ) *approximation-stability with respect to the k-median objective, each cluster in* C_T *has size at least* $2n$*, and* $C_T = C^*$*. Then,*

1. *Fewer than n points* $\mathbf{a}_i$ *have* $w_2(\mathbf{a}_i) - w(\mathbf{a}_i) \leq (c-1)w_{avg}/\epsilon$.
2. *At most* $5\epsilon n/(c-1)$ *points* $\mathbf{a}_i$ *have* $w(\mathbf{a}_i) \geq (c-1)w_{avg}/(5\epsilon)$.

Proof: For part (1), suppose that *n* points $\mathbf{a}_i$ have $w_2(\mathbf{a}_i) - w(\mathbf{a}_i) \leq (c-1)w_{avg}/\epsilon$. Consider modifying C_T to a new clustering C^0 by moving each of these points $\mathbf{a}_i$ into the cluster containing its second-closest center. By assumption, the *k*-means cost of the clustering has increased by at most $\epsilon n(c-1)w_{avg}/\epsilon = (c-1) \cdot OPT$. This means that the cost of the new clustering is at most *cOPT*. However, $dist(\mathcal{C}', \mathcal{C}_T) = \epsilon$ because (a) we moved *n* points to different clusters, and (b) each cluster in C_T has size at least 2*n* so the optimal permutation σ in the definition of *dist* remains the identity. So, this contradicts approximation stability. Part (2) follows from the definition of "average"; if it did not hold then $\sum_{i=1}^{n} w(\mathbf{a}_i) > nw_{avg}$, a contradiction. ■

A datapoint $\mathbf{a}_i$ is *bad* if it satisfies either item (1) or (2) of Lemma 7.6 and *good* if it satisfies neither one. So, there are at most $b = \epsilon n + \frac{5\epsilon n}{c-1}$ bad points and the rest are good.

Define "critical distance" $d_{crit} = \frac{(c-1)w_{avg}}{5\epsilon}$. Lemma 7.6 implies that the good points have distance at most d_{crit} to the center of their own cluster in C^* and distance at least $5d_{crit}$ to the center of any other cluster in C^*.

This suggests the following algorithm. Suppose we create a graph *G* with the points $\mathbf{a}_i$ as vertices, and edges between any two points $\mathbf{a}_i$ and $\mathbf{a}_j$ with $d(\mathbf{a}_i, \mathbf{a}_j) < 2d_{crit}$. Notice that by triangle inequality, the good points within the same cluster in C^* have distance less than $2d_{crit}$ from each other so they will be fully connected and form a clique. Also, again by triangle inequality, any edge that goes between different clusters must be between two bad points. In particular, if $\mathbf{a}_i$ is a good point in one cluster, and it has an edge to some other point $\mathbf{a}_j$, then $\mathbf{a}_j$ must have distance less than $3d_{crit}$ to the center of $\mathbf{a}_i$'s cluster. This means that if $\mathbf{a}_j$ had a different closest center, which obviously would also be at distance less than $3d_{crit}$, then $\mathbf{a}_i$ would have distance less than $2d_{crit} + 3d_{crit} = 5d_{crit}$ to that center, violating its goodness. So, bridges in *G* between different clusters can only occur between bad points.

Assume now that each cluster in C_T has size at least $2b+1$; this is the sense in which we are requiring that *n* be small compared to the smallest cluster in C_T. In this case, create a new graph *H* by connecting any two points $\mathbf{a}_i$ and $\mathbf{a}_j$ that share at least $b+1$ neighbors in common in *G*, themselves included. Since every cluster has at least $2b + 1 - b = b + 1$ good points, and these points are fully connected in *G*, this means that *H* will contain an edge between every pair of good points in the same cluster. On the other hand, since the only edges in *G* between different clusters are between bad points, and there are at most *b* bad points, this means that *H* will not have any edges between different clusters in C_T. Thus, if we take the *k* largest connected components in *H*, these will all correspond to subsets of different clusters in C_T, with at most *b* points remaining.

At this point we have a correct clustering of all but at most *b* points in *A*. Call these clusters $C_1, \ldots, C_k$, where $C_j \subseteq C_j^*$. To cluster the remaining points $\mathbf{a}_i$, we assign them to the

cluster C_j that minimizes the median distance between $\mathbf{a}_i$ and points in C_j. Since each C_j has more good points than bad points, and each good point in C_j has distance at most d_{crit} to center $\mathbf{c}^*_j$, by triangle inequality the median of these distances must lie in the range $[d(\mathbf{a}_i, \mathbf{c}^*_i) - d_{crit}, d(\mathbf{a}_i, \mathbf{c}^*_i) + d_{crit}]$. This means that this second step will correctly cluster all points $\mathbf{a}_i$ for which $w_2(\mathbf{a}_i) - w(\mathbf{a}_i) > 2d_{crit}$. In particular, we correctly cluster all points except possibly for some of the at most n satisfying item (1) of Lemma 7.6.

The above discussion assumes the value d_{crit} is known to our algorithm; we leave it as an exercise to the reader to modify the algorithm to remove this assumption. Summarizing, we have the following algorithm and theorem.

Algorithm *k*-**Median Stability** (given c, ϵ, d_{crit})

1. Create a graph G with a vertex for each datapoint in A, and an edge between vertices i and j if $d(\mathbf{a}_i, \mathbf{a}_j) \leq 2d_{crit}$.
2. Create a graph H with a vertex for each vertex in G and an edge between vertices i and j if i and j share at least $b + 1$ neighbors in common, themselves included, for $b = \epsilon n + \frac{5\epsilon n}{c-1}$. Let $C_1, \ldots, C_k$ denote the k largest connected components in H.
3. Assign each point not in $C_1 \cup \ldots \cup C_k$ to the cluster C_j of smallest median distance.

Theorem 7.7 *Assume A satisfies (c, ϵ) approximation-stability with respect to the kmedian objective, that each cluster in C_T has size at least $\frac{10\epsilon}{c-1}n + 2\epsilon n + 1$, and that $\mathrm{C}_T = \mathrm{C}^*$. Then Algorithm k-*Median Stability *will find a clustering* C *such that* $dist(\mathcal{C}, \mathcal{C}_T) \leq \epsilon$.

7.7 High-Density Clusters

We now turn from the assumption that clusters are center-based to the assumption that clusters consist of high-density regions, separated by low-density moats such as in Figure 7.1.

7.7.1 Single Linkage

One natural algorithm for clustering under the high-density assumption is called *single linkage*. This algorithm begins with each point in its own cluster and then repeatedly merges the two "closest" clusters into one, where the distance between two clusters is defined as the *minimum* distance between points in each cluster. That is, $d_{min}(C, C^0) = \min_{\mathbf{x} \in C, \mathbf{y} \in C^0} d(\mathbf{x}, \mathbf{y})$, and the algorithm merges the two clusters C and C^0 whose d_{min} value is smallest over all pairs of clusters breaking ties arbitrarily. It then continues until there are only k clusters. This is called an *agglomerative* clustering algorithm because it begins with many clusters and then starts merging, or agglomerating them together. [36]

[36] Other agglomerative algorithms include *complete linkage* which merges the two clusters whose *maximum* distance between points is smallest, and Ward's algorithm described earlier that merges the two clusters that cause the k-means cost to increase by the least.

Singlelinkage is equivalent to running Kruskal's minimum-spanning-tree algorithm, but halting when there are k trees remaining. The following theorem is fairly immediate.

Theorem 7.8 *Suppose the desired clustering $C_1^*, \ldots, C_k^*$ satisfies the property that there exists some distance σ such that*

1. *any two data points in different clusters have distance at least σ, and*
2. *for any cluster C_i^* and any partition of C_i^* into two non-empty sets A and $C_i^* \setminus A$, there exist points on each side of the partition of distance less than σ.*

Then, single-linkage will correctly recover the clustering $C_1^*, \ldots, C_k^*$.

Proof: Consider running the algorithm until all pairs of clusters C and C^0 have $d_{min}(C,C^0) \geq \sigma$. At that point, by (2), each target cluster C_i^* will be fully contained within some cluster of the single-linkage algorithm. On the other hand, by (1) and by induction, each cluster C of the single-linkage algorithm will be fully contained within some C_i^* of the target clustering, since any merger of subsets of distinct target clusters would require $d_{min} \geq \sigma$. Therefore, the single-linkage clusters are indeed the target clusters. ■

7.7.2 Robust Linkage

The single-linkage algorithm is fairly brittle. A few points bridging the gap between two different clusters can cause it to do the wrong thing. As a result, there has been significant work developing more robust versions of the algorithm.

One commonly used robust version of single linkage is Wishart's algorithm. A ball of radius r is created for each point with the point as center. The radius r is gradually increased starting from $r = 0$. The algorithm has a parameter t. When a ball has t or more points the center point becomes active. When two balls with active centers intersect the two center points are connected by an edge. The parameter t prevents a thin string of points between two clusters from causing a spurious merger. Note that Wishart's algorithm with $t = 1$ is the same as single linkage.

In fact, if one slightly modifies the algorithm to define a point to be live if its ball of radius $r/2$ contains at least t points, then it is known [CD10] that a value of $t = O(d\log n)$ is sufficient to recover a nearly correct solution under a natural distributional formulation of the clustering problem. Specifically, suppose data points are drawn from some probability distribution D over R^d, and that the clusters correspond to high-density regions surrounded by lower-density moats. More specifically, the assumption is that

1. for some distance $\sigma > 0$, the σ-interior of each target cluster C_i^* has density at least some quantity λ (the σ-interior is the set of all points at distance at least σ from the boundary of the cluster),
2. the region between target clusters has density less than $\lambda(1-\epsilon)$ for some $\epsilon > 0$,
3. the clusters should be separated by distance greater than 2σ, and

4. the σ-interior of the clusters contains most of their probability mass.

Then, for sufficiently large n, the algorithm will with high probability find nearly correct clusters. In this formulation, we allow points in low-density regions that are not in any target clusters at all. For details, see [CD10].

Robust Median Neighborhood Linkage robustifies single linkage in a different way. This algorithm guarantees that if it is possible to delete a small fraction of the data such that for all remaining points x, most of their $|C^*(x)|$ nearest neighbors indeed belong to their own cluster $C^*(x)$, then the hierarchy on clusters produced by the algorithm will include a close approximation to the true clustering. We refer the reader to [BLG14] for the algorithm and proof.

7.8 Kernel Methods

Kernel methods combine aspects of both center-based and density-based clustering. In center-based approaches like k-means or k-center, once the cluster centers are fixed, the Voronoi diagram of the cluster centers determines which cluster each data point belongs to. This implies that clusters are pairwise linearly separable.

If we believe that the true desired clusters may not be linearly separable, and yet we wish to use a center-based method, then one approach, as in the chapter on learning, is to use a kernel. Recall that a kernel function $K(\mathbf{x},\mathbf{y})$ can be viewed as performing an implicit mapping φ of the data into a possibly much higher dimensional space, and then taking a dot-product in that space. That is, $K(\mathbf{x},\mathbf{y}) = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{y})$. This is then viewed as the affinity between points $\mathbf{x}$ and $\mathbf{y}$. We can extract distances in this new space using the equation $|\mathbf{z_1} - \mathbf{z_2}|^2 = \mathbf{z_1} \cdot \mathbf{z_1} + \mathbf{z_2} \cdot \mathbf{z_2} - 2\mathbf{z_1} \cdot \mathbf{z_2}$, so in particular we have $|\varphi(\mathbf{x})-\varphi(\mathbf{y})|^2 = K(\mathbf{x},\mathbf{x})+K(\mathbf{y},\mathbf{y})-2K(\mathbf{x},\mathbf{y})$. We can then run a center-based clustering algorithm on these new distances.

One popular kernel function to use is the Gaussian kernel. The Gaussian kernel uses an affinity measure that emphasizes closeness of points and drops off exponentially as the points get farther apart. Specifically, we define the affinity between points $\mathbf{x}$ and $\mathbf{y}$ by

$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{y}\|^2}.$$

Another way to use affinities is to put them in an affinity matrix, or weighted graph. This graph can then be separated into clusters using a graph partitioning procedure such as the one in following section.

7.9 Recursive Clustering based on Sparse Cuts

We now consider the case that data are nodes in an undirected connected graph $G(V,E)$ where an edge indicates that the end point vertices are similar. Recursive clustering starts with all vertices in one cluster and recursively splits a cluster into two parts whenever

there is a small number of edges from one part to the other part of the cluster. Formally, for two disjoint sets S and T of vertices, define

$$\Phi(S,T) = \frac{\text{Number of edges from } S \text{ to } T}{\text{Total number of edges incident to } S \text{ in } G}.$$

$\Phi(S,T)$ measures the relative strength of similarities between S and T. Let $d(i)$ be the degree of vertex i and for a subset S of vertices, let $d(S) = \sum_{i \in S} d(i)$. Let m be the total number of edges in the graph. The following algorithm aims to cut only a small fraction of the edges and to produce clusters that are internally consistent in that no subset of the cluster has low similarity to the rest of the cluster.

Recursive Clustering: Select an appropriate value for ε. If a current cluster W has a subset S with $d(S) \leq \frac{1}{2} d(W)$ and $\Phi(S, S \subseteq W) \leq \varepsilon$, then split W into two clusters S and $W \setminus S$. Repeat until no such split is possible.

Theorem 7.9 *At termination of Recursive Clustering, the total number of edges between vertices in different clusters is at most* $O(\varepsilon m \ln n)$.

Proof: Each edge between two different clusters at the end was deleted at some stage by the algorithm. We will "charge" edge deletes to vertices and bound the total charge. When the algorithm partitions a cluster W into S and $W \setminus S$ with $d(S) \leq (1/2)d(W)$, each $k \in S$ is charged $\frac{d(k)}{d(W)}$ times the number of edges being deleted. Since $\Phi(S, W \setminus S) \leq \varepsilon$, the charge added to each $k \in W$ is a most $\varepsilon d(k)$. A vertex is charged only when it is in the smaller part, $d(S) \leq d(W)/2$, of the cut. So between any two times it is charged, $d(W)$ is reduced by a factor of at least two and so a vertex can be charged at most $\log_2 m \leq O(\ln n)$ times, proving the theorem. ■

Implementing the algorithm requires computing $\text{Min}_{S \subseteq W} \Phi(S, W \setminus S)$ which is an NPhard problem. So the theorem cannot be implemented right away. Luckily, eigenvalues and eigenvectors, which can be computed fast, give an approximate answer. The connection between eigenvalues and sparsity, known as Cheeger's inequality, is deep with applications to Markov chains among others. We do not discuss this here.

7.10 Dense Submatrices and Communities

Represent n data points in d-space by the rows of an $n \times d$ matrix A. Assume that A has all nonnegative entries. Examples to keep in mind for this section are the documentterm matrix and the customer-product matrix. We address the question of how to define and find efficiently a coherent large subset of rows. To this end, the matrix A can be represented by a bipartite graph Figure 7.5. One side has a vertex for each row and the other side a vertex for each column. Between the vertex for row i and the vertex for column j, there is an edge with weight a_{ij}.

We want a subset S of row vertices and a subset T of column vertices so that

$$A(S,T) = \sum_{i \in S, j \in T} a_{ij}$$

is high. This simple definition is not good since $A(S,T)$ will be maximized by taking all rows and columns. We need a balancing criterion that ensures that $A(S,T)$ is high relative to the sizes of S and T. One possibility is to maximize $\frac{A(S,T)}{|S||T|}$. This is not a good measure either, since it is maximized by the single edge of highest weight. The definition we use is the following. Let A be a matrix with nonnegative entries. For a subset S of rows and a subset T of columns, the *density* $d(S,T)$ of S and T is $d(S,T) = \frac{A(S,T)}{\sqrt{|S||T|}}$. The *density* $d(A)$ of A is defined as the maximum value of $d(S,T)$ over all subsets of rows and columns. This definition applies to bipartite as well as non bipartite graphs.

One important case is when A's rows and columns both represent the same set and a_{ij} is the similarity between object i and object j. Here $d(S,S) = \frac{A(S,S)}{|S|}$. If A is an $n \times n$ 0-1 matrix, it can be thought of as the adjacency matrix of an undirected graph, and $d(S,S)$ is the average degree of a vertex in S. The subgraph of maximum average degree in a graph can be found exactly by network flow techniques, as we will show in the next

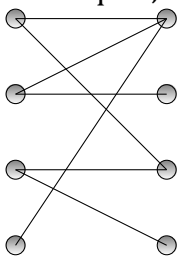


Figure 7.5: Example of a bipartite graph.

section. We do not know an efficient (polynomial-time) algorithm for finding $d(A)$ exactly in general. However, we show that $d(A)$ is within a $O(\log^2 n)$ factor of the top singular value of A assuming $|a_{ij}| \leq 1$ for all i and j. This is a theoretical result. The gap may be much less than $O(\log^2 n)$ for many problems, making singular values and singular vectors quite useful. Also, S and T with $d(S,T) \geq \Omega(d(A)/\log^2 n)$ can be found algorithmically. **Theorem 7.10** *Let A be an n × d matrix with entries between 0 and 1. Then*

$$\sigma_1(A) \geq d(A) \geq \frac{\sigma_1(A)}{4 \log n \log d}.$$

Furthermore, subsets S and T satisfying $d(S,T) \geq \frac{\sigma_1(A)}{4 \log n \log d}$ *may be found from the top singular vector of A.*

Proof: Let S and T be the subsets of rows and columns that achieve $d(A) = d(S,T)$. Consider an n-vector $\mathbf{u}$ that is $\frac{1}{\sqrt{|S|}}$ on S and 0 elsewhere and a d-vector $\mathbf{v}$ that is $\frac{1}{\sqrt{|T|}}$ on T and 0 elsewhere. Then,

$$\sigma_1(A) \geq \mathbf{u}^T A \mathbf{v} = \sum_{ij} u_i v_j a_{ij} = d(S,T) = d(A)$$

establishing the first inequality.

To prove the second inequality, express $\sigma_1(A)$ in terms of the first left and right singular vectors $\mathbf{x}$ and $\mathbf{y}$.

$$\sigma_1(A) = \mathbf{x}^T A \mathbf{y} = \sum_{i,j} x_i a_{ij} y_j, \qquad |\mathbf{x}| = |\mathbf{y}| = 1.$$

Since the entries of A are nonnegative, the components of the first left and right singular vectors must all be nonnegative, that is, $x_i \geq 0$ and $y_j \geq 0$ for all i and j. To bound $\sum_{i,j} x_i a_{ij} y_j$, break the summation into $O(\log n \log d)$ parts. Each part corresponds to a given α and β and consists of all i such that $\alpha \leq x_i < 2\alpha$ and all j such that $\beta \leq y_i < 2\beta$. The $\log n \log d$ parts are defined by breaking the rows into $\log n$ blocks with α equal to $\frac{1}{2}\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, 2\frac{1}{\sqrt{n}}, 4\frac{1}{\sqrt{n}}, \cdots, 1$ and by breaking the columns into $\log d$ blocks with β equal to $\frac{1}{2}\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}, \frac{2}{\sqrt{d}}, \frac{4}{\sqrt{d}}, \cdots, 1$. The i such that $x_i < \frac{1}{2\sqrt{n}}$ and the j such that $y_j < \frac{1}{2\sqrt{d}}$ will be ignored at a loss of at most $\frac{1}{4}\sigma_1(A)$. Exercise 7.27 proves the loss is at most this amount.

Since $\sum_i x_i^2 = 1$, the set $S = \{i | \alpha \leq x_i < 2\alpha\}$ has $|S| \leq \frac{1}{\alpha^2}$ and similarly, $T = \{j | \beta \leq y_j \leq 2\beta\}$ has $|T| \leq \frac{1}{\beta^2}$. Thus

$$\begin{aligned}
\sum_{\alpha \leq x_i \leq 2\alpha} \sum_{\beta \leq y_j \leq 2\beta} x_i y_j a_{ij} &\leq 4\alpha\beta A(S,T) \\
&\leq 4\alpha\beta d(S,T)\sqrt{|S||T|} \\
&\leq 4d(S,T) \\
&\leq 4d(A).
\end{aligned}$$

From this it follows that

$$\sigma_1(A) \leq 4d(A)\log n \log d$$

or

$$d(A) \geq \frac{\sigma_1(A)}{4 \log n \log d}$$

proving the second inequality.

It is clear that for each of the values of (α,β), we can compute $A(S,T)$ and $d(S,T)$ as above and taking the best of these $d(S,T)$'s gives us an algorithm as claimed in the theorem. ■

Note that in many cases, the nonzero values of x_i and y_j after zeroing out the low entries will only go from $\frac{1}{2}\frac{1}{\sqrt{n}}$ to $\frac{c}{\sqrt{n}}$ for x_i and $\frac{1}{2}\frac{1}{\sqrt{d}}$ to $\frac{c}{\sqrt{d}}$ for y_j, since the singular vectors are likely to be balanced given that a_{ij} are all between 0 and 1. In this case, there will be $O(1)$ groups only and the log factors disappear.

Another measure of density is based on similarities. Recall that the similarity between objects represented by vectors (rows of A) is defined by their dot products. Thus, similarities are entries of the matrix AA^T. Define the average cohesion $f(S)$ of a set S of rows of A to be the sum of all pairwise dot products of rows in S divided by $|S|$. The average cohesion of A is the maximum over all subsets of rows of the average cohesion of the subset.

Since the singular values of AA^T are squares of singular values of A, we expect $f(A)$ to be related to $\sigma_1(A)^2$ and $d(A)^2$. Indeed it is. We state the following without proof.

Lemma 7.11 $d(A)^2 \leq f(A) \leq d(A)\log n$. *Also,* $\sigma_1(A)^2 \geq f(A) \geq \frac{c\sigma_1(A)^2}{\log n}$.

$f(A)$ can be found exactly using flow techniques as we will see later.

7.11 Community Finding and Graph Partitioning

Assume that data are nodes in a possibly weighted graph where edges represent some notion of affinity between their endpoints. In particular, let $G = (V,E)$ be a weighted graph. Given two sets of nodes S and T, define

$$E(S,T) = \sum_{i\in S\, j\in T} e_{ij}.$$

We then define the *density* of a set S to be

$$d(S,S) = \frac{E(S,S)}{|S|}.$$

If G is an undirected graph, then $d(S,S)$ can be viewed as the average degree in the vertex-induced subgraph over S. The set S of maximum density is therefore the subgraph of maximum average degree. Finding such a set can be viewed as finding a tight-knit community inside some network. In the next section, we describe an algorithm for finding such a set using network flow techniques.

Flow Methods Here we consider dense induced subgraphs of a graph. An induced subgraph of a graph consisting of a subset of the vertices of the graph along with all edges of the graph that connect pairs of vertices in the subset of vertices. We show that finding an induced subgraph with maximum average degree can be done by network flow

techniques. This is simply maximizing the density $d(S,S)$ over all subsets S of the graph. First consider the problem of finding a subset of vertices such that the induced subgraph has average degree at least λ for some parameter λ. Then do a binary search on the value of λ until the maximum λ for which there exists a subgraph with average degree at least λ is found.

Given a graph G in which one wants to find a dense subgraph, construct a directed graph H from the given graph and then carry out a flow computation on H. H has a node for each edge of the original graph, a node for each vertex of the original graph, plus two additional nodes s and t. There is a directed edge with capacity one from s to each node corresponding to an edge of the original graph and a directed edge with infinite capacity from each node corresponding to an edge of the original graph to the two nodes corresponding to the vertices the edge connects. Finally, there is a directed edge with capacity λ from each node corresponding to a vertex of the original graph to t.

Notice there are three types of cut sets of the directed graph that have finite capacity, Figure 7.6. The first cuts all arcs from the source. It has capacity e, the number of edges of the original graph. The second cuts all edges into the sink. It has capacity λv, where v is the number of vertices of the original graph. The third cuts some arcs from s and some arcs into t. It partitions the set of vertices and the set of edges of the original graph into two blocks. The first block contains the source node s, a subset of the edges e_s, and a

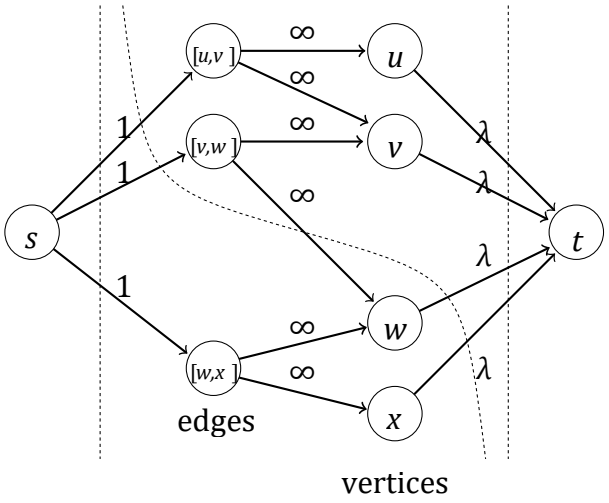


Figure 7.6: The directed graph H used by the flow technique to find a dense subgraph

subset of the vertices v_s defined by the subset of edges. The first block must contain both end points of each edge in e_s; otherwise an infinite arc will be in the cut. The second block contains t and the remaining edges and vertices. The edges in this second block either connect vertices in the second block or have one endpoint in each block. The cut set will cut some infinite arcs from edges not in e_s coming into vertices in v_s. However, these arcs are directed from nodes in the block containing t to nodes in the block containing s. Note that any finite capacity cut that leaves an edge node connected to s must cut the two

related vertex nodes from t, Figure 7.6. Thus, there is a cut of capacity $e - e_s + \lambda v_s$ where v_s and e_s are the vertices and edges of a subgraph. For this cut to be the minimal cut, the quantity $e - e_s + \lambda v_s$ must be minimal over all subsets of vertices of the original graph and the capcity must be less than e and also less than λv.

If there is a subgraph with v_s vertices and e_s edges where the ratio $\frac{e_s}{v_s}$ is sufficiently large so that $\frac{e_S}{v_S} > \frac{e}{v}$, then for $\lambda \text{ such that } \frac{e_S}{v_S} > \lambda > \frac{e}{v}$, $e_s - \lambda v_s > 0$ and $e - e_s + \lambda v_s < e$.
Similarly $e < \lambda v$ and thus $e - e_s + \lambda v_s < \lambda v$. This implies that the cut $e - e_s + \lambda v_s$ is less than either e or λv and the flow algorithm will find a nontrivial cut and hence a proper subset. For different values of λ in the above range there maybe different nontrivial cuts.

Note that for a given density of edges, the number of edges grows as the square of the number of vertices and $\frac{e_s}{v_s}$ is less likely to exceed $\frac{e}{v}$ if v_S is small. Thus, the flow method works well in finding large subsets since it works with $\frac{e_S}{v_S}$. To find small communities one would need to use a method that worked with $\frac{e_S}{v_S^2}$ as the following example illustrates.

Example: Consider finding a dense subgraph of 1,000 vertices and 2,000 internal edges in a graph of 10^6 vertices and 6×10^6 edges. For concreteness, assume the graph was generated by the following process. First, a 1,000-vertex graph with 2,000 edges was generated as a

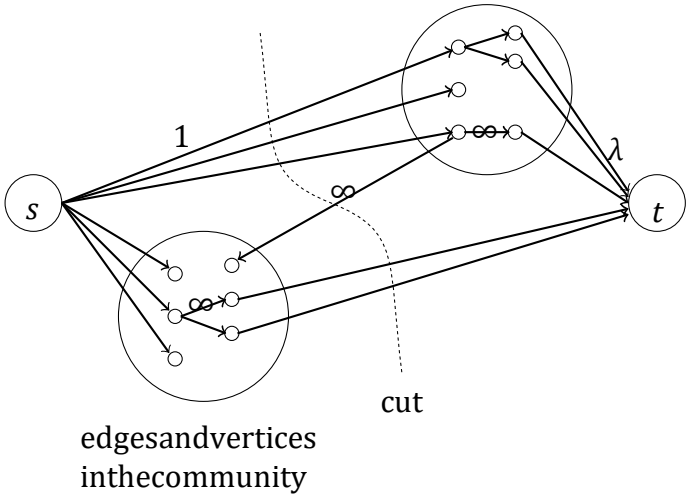


Figure 7.7: Cut in flow graph

random regular degree four graph. The 1,000-vertex graph was then augmented to have 10^6 vertices and edges were added at random until all vertices were of degree 12. Note that each vertex among the first 1,000 has four edges to other vertices among the first 1,000 and eight edges to other vertices. The graph on the 1,000 vertices is much denser than the whole graph in some sense. Although the subgraph induced by the 1,000 vertices has four edges per vertex and the full graph has twelve edges per vertex, the probability

of two vertices of the 1,000 being connected by an edge is much higher than for the graph as a whole. The probability is given by the ratio of the actual number of edges connecting vertices among the 1,000 to the number of possible edges if the vertices formed a complete graph.

$$p = \frac{e}{\binom{v}{2}} = \frac{2e}{v(v-1)}$$

For the 1,000 vertices, this number is $p = \frac{2\times 2{,}000}{1{,}000\times 999} \cong 4 \times 10^{-3}$. For the entire graph this number is $p = \frac{2\times 6\times 10^6}{10^6\times 10^6} = 12 \times 10^{-6}$. This difference in probability of two vertices being connected should allow us to find the dense subgraph. ■

In our example, the cut of all arcs out of s is of capacity 6×10^6, the total number of edges in the graph, and the cut of all arcs into t is of capacity λ times the number of vertices or $\lambda \times 10^6$. A cut separating the 1,000 vertices and 2,000 edges would have capacity $6\times10^6 - 2{,}000 + \lambda\times 1{,}000$. This cut cannot be the minimum cut for any value of λ since $\frac{e_s}{v_s} = 2$ and $\frac{e}{v} = 6$, hence $\frac{e_s}{v_s} < \frac{e}{v}$. The point is that to find the 1,000 vertices, we have to maximize $A(S,S)/|S|^2$ rather than $A(S,S)/|S|$. Note that $A(S,S)/|S|^2$ penalizes large |S| much more and therefore can find the 1,000 node "dense" subgraph.

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad V = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

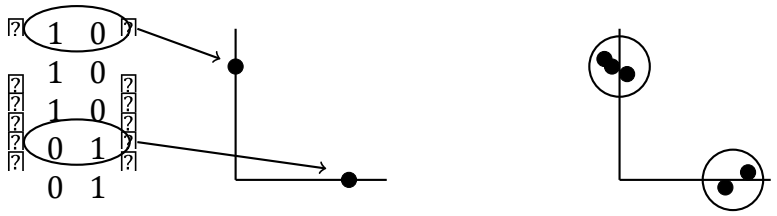


Figure 7.8: Illustration of spectral clustering.

7.12 Spectral clustering applied to social networks

Finding communities in social networks is different from other clustering for several reasons. First we often want to find communities of size say 20 to 50 in networks with 100 million vertices. Second a person is in a number of overlapping communities and thus we are not finding disjoint clusters. Third there often are a number of levels of structure and a set of dominant communities may be hiding a set of weaker communities that are of interest. Spectral clustering is one approach to these issues.

In spectral clustering of the vertices of a graph, one creates a matrix V whose columns correspond to the first k singular vectors of the adjacency matrix. Each row of V is the projection of a row of the adjacency matrix to the space spanned by the k singular vectors. In the example below, the graph has five vertices divided into two cliques, one consisting

of the first three vertices and the other the last two vertices. The top two right singular vectors of the adjacency matrix, not normalized to length one, are $(1,1,1,0,0)^T$ and $(0,0,0,1,1)^T$. The five rows of the adjacency matrix projected to these vectors form the 5×2 matrix in Figure 7.8. Here, there are two ideal clusters with all edges inside a cluster being present including self-loops and all edges between clusters being absent. The five rows project to just two points, depending on which cluster the rows are in. If the clusters were not so ideal and instead of the graph consisting of two disconnected cliques, the graph consisted of two dense subsets of vertices where the two sets were connected by only a few edges, then the singular vectors would not be indicator vectors for the clusters but close to indicator vectors. The rows would be mapped to two clusters of points instead of two points. A *k*-means clustering algorithm would find the clusters.

If the clusters were overlapping, then instead of two clusters of points, there would be three clusters of points where the third cluster corresponds to the overlapping vertices of the two clusters. Instead of using *k*-means clustering, we might instead find the minimum 1-norm vector in the space spanned by the two singular vectors. The minimum 1-norm vector will not be an indicator vector, so we would threshold its values to create an indicator vector for a cluster. Instead of finding the minimum 1-norm vector in the space spanned by the singular vectors in *V,* we might look for a small 1-norm vector close to the subspace.

$$\min(1 - |\mathbf{x}|_1 + \alpha\cos(\theta))_{\mathbf{x}}$$

Here θ is the cosine of the angle between $\mathbf{x}$ and the space spanned by the two singular vectors. α is a control parameter that determines how close we want the vector to be to the subspace. When α is large, $\mathbf{x}$ must be close to the subspace. When α is zero, $\mathbf{x}$ can be anywhere.

Finding the minimum 1-norm vector in the space spanned by a set of vectors can be formulated as a linear programming problem. To find the minimum 1-norm vector in *V,* write $V\,\mathbf{x} = \mathbf{y}$ where we want to solve for both $\mathbf{x}$ and $\mathbf{y}$. Note that the format is different from the usual format for a set of linear equations $A\mathbf{x} = \mathbf{b}$ where $\mathbf{b}$ is a known vector.

Finding the minimum 1-norm vector looks like a nonlinear problem.

$$\min|\mathbf{y}|_1 \text{ subject to } V\,\mathbf{x} = \mathbf{y}$$

To remove the absolute value sign, write $\mathbf{y} = \mathbf{y_1} - \mathbf{y_2}$ with $\mathbf{y_1} \geq 0$ and $\mathbf{y_2} \geq 0$. Then solve

$$\min\left(\sum_{i=1}^{n} y_{1i} + \sum_{i=1}^{n} y_{2i}\right) \text{ subject to } V\,\mathbf{x} = \mathbf{y},\ \mathbf{y_1} \geq 0, \text{ and } \mathbf{y}_2 \geq 0.$$

Write $V\,\mathbf{x} = \mathbf{y_1} - \mathbf{y_2}$ as $V\,\mathbf{x} - \mathbf{y_1} + \mathbf{y_2} = 0$. then we have the linear equations in a format we are accustomed to.

$$[V, -I, I] \begin{pmatrix} \mathbf{x} \\ \mathbf{y_1} \\ \mathbf{y_2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

This is a linear programming problem. The solution, however, happens to be $\mathbf{x} = 0$, $\mathbf{y_1} = 0$, and $\mathbf{y_2} = 0$. To resolve this, add the equation $y_{1i} = 1$ to get a community containing the vertex *i*.

Often we are looking for communities of 50 or 100 vertices in graphs with hundreds of million of vertices. We want a method to find such communities in time proportional to the size of the community and not the size of the entire graph. Here spectral clustering can be used but instead of calculating singular vectors of the entire graph, we do something else. Consider a random walk on a graph. If we walk long enough the probability distribution converges to the first eigenvector. However, if we take only a few steps from a start vertex or small group of vertices that we believe define a cluster, the probability will distribute over the cluster with some of the probability leaking out to the remainder of the graph. To get the early convergence of several vectors that ultimately converge to the first few singular vectors, take a subspace $[\mathbf{x}, A\mathbf{x}, A^2\mathbf{x}, A^3\mathbf{x}]$ and propagate the subspace. At each iteration find an orthonormal basis and then multiply each basis vector by *A*. Then take the resulting basis vectors after a few steps, say five, and find a minimum 1-norm vector in the subspace.

A third issue that arises is when a dominant structure hides an important weaker structure. One can run their algorithm to find the dominant structure and then weaken the dominant structure by randomly removing edges in the clusters so that the edge density is similar to the remainder of the network. Then reapplying the algorithm often will uncover weaker structure. Real networks often have several levels of structure. The technique can also be used to improve state of the art clustering algorithms. After weakening the dominant structure to find the weaker hidden structure one can go back to the original data and weaken the hidden structure and reapply the algorithm to again find the dominant structure. This improves most state of the art clustering algorithms.

7.13 Bibliographic Notes

Clustering has a long history. For a good general survey, see [Jai10]. For a collection of surveys giving an in-depth treatment of many different approaches to, applications of, and perspectives on clustering, see [HMMR15]; e.g., see [AB15] for a good discussion of center-based clustering. Lloyd's algorithm for k-means clustering is from [Llo82], and The k-means++ initialization method is due to Arthur and Vassilvitskii [AV07]. Ward's algorithm, from 1963, appears in [War63]. Analysis of the farthest-traversal algorithm for the k-center problem is due to Gonzalez [Gon85].

Theorem 7.4 is from [KV09]. Analysis of spectral clustering in stochastic models is given in [McS01], and the analysis of spectral clustering without a stochastic model and 7.5.2 is due to [KK10].

The definition of approximation-stability is from [BBG13] and [BBV08], and the analysis given in Section 7.6 is due to [BBG13].

Single-linkage clustering goes back at least to Florek et al. [FL P+51], and Wishart's robust version is from [Wis69]. Extensions of Theorem 7.8 are given in [BBV08], and theoretical guarantees for different forms of robust linkage are given in [CD10] and [BLG14].

A good survey of kernel methods in clustering appears in [FCMR08].

Section 7.9 is a simplified version of [KVV04]. Section 7.10 is from [RV99].

7.14 Exercises

Exercise 7.1 *Construct examples where using distances instead of distance squared gives bad results for Gaussian densities. For example, pick samples from two 1-dimensional unit variance Gaussians, with their centers 10 units apart. Cluster these samples by trial and error into two clusters, first according to k-means and then according to the k-median criteria. The k-means clustering should essentially yield the centers of the Gaussians as cluster centers. What cluster centers do you get when you use the k-median criterion?*

Exercise 7.2 *Let* $v = (1,3)$*. What is the* L_1 *norm of v? The* L_2 *norm? The square of the* L_1 *norm?*

Exercise 7.3 *Show that in 1-dimension, the center of a cluster that minimizes the sum of distances of data points to the center is in general not unique. Suppose we now require the center also to be a data point; then show that it is the median element (not the mean). Further in 1-dimension, show that if the center minimizes the sum of squared distances to the data points, then it is unique.*

Exercise 7.4 *Construct a block diagonal matrix A with three blocks of size 50. Each matrix element in a block has value* $p = 0.7$ *and each matrix element not in a block has value* $q = 0.3$*. Generate a* 150×150 *matrix B of random numbers in the range [0,1]. If* $b_{ij} \geq a_{ij}$ *replace* a_{ij} *with the value one. Otherwise replace* a_{ij} *with value zero. The rows of A have three natural clusters. Generate a random permutation and use it to permute the rows and columns of the matrix A so that the rows and columns of each cluster are randomly distributed.*

1. *Apply the k-means algorithm to A with* $k = 3$*. Do you find the correct clusters?*
2. *Apply the k-means algorithm to A for* $1 \leq k \leq 10$*. Plot the value of the sum of squares to the cluster centers versus k. Was three the correct value for k*?

Exercise 7.5 *Let M be a* $k \times k$ *matrix whose elements are numbers in the range [0,1]. A matrix entry close to one indicates that the row and column of the entry correspond to closely related items and an entry close to zero indicates unrelated entities. Develop an algorithm to match each row with a closely related column where a column can be matched with only one row.*

Exercise 7.6 *The simple greedy algorithm of Section 7.3 assumes that we know the clustering radius r. Suppose we do not. Describe how we might arrive at the correct r?*

Exercise 7.7 *For the k-median problem, show that there is at most a factor of two ratio between the optimal value when we either require all cluster centers to be data points or allow arbitrary points to be centers.*

Exercise 7.8 *For the k-means problem, show that there is at most a factor of four ratio between the optimal value when we either require all cluster centers to be data points or allow arbitrary points to be centers.*

Exercise 7.9 *Consider clustering points in the plane according to the k-median criterion, where cluster centers are required to be data points. Enumerate all possible clustering's and*

select the one with the minimum cost. The number of possible ways of labeling n points, each with a label from {1,2,...,k} is k^n which is prohibitive. Show that we can find the optimal clustering in time at most a constant times $\binom{n}{k} + k^2$. Note that $\binom{n}{k} \leq n^k$ which is much smaller than k^n when k << n.

Exercise 7.10 *Suppose in the previous exercise, we allow any point in space (not necessarily data points) to be cluster centers. Show that the optimal clustering may be found in time at most a constant times n^{2k_2}.*

Exercise 7.11 *Corollary 7.2 shows that for a set of points $\{a_1, a_2, ..., a_n\}$, there is a unique point x, namely their centroid, which minimizes $\sum_{i=1}^{n} |a_i - x|^2$. Show examples*

where the x minimizing $\sum_{i=1}^{n} |a_i - x|$ is not unique. (Consider just points on the real line.)
Show examples where the x defined as above are far apart from each other.

Exercise 7.12 *Let $\{\mathbf{a_1}, \mathbf{a_2}, ..., \mathbf{a_n}\}$ be a set of unit vectors in a cluster. Let $\mathbf{c} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{a_i}$ be the cluster centroid. The centroid $\mathbf{c}$ is not in general a unit vector. Define the similarity between two points $\mathbf{a}_i$ and $\mathbf{a}_j$ as their dot product. Show that the average cluster similarity $\frac{1}{n^2} \sum_{i,j} \mathbf{a_i} \mathbf{a_j}^T$ is the same whether it is computed by averaging all pairs or computing the average similarity of each point with the centroid of the cluster.*

Exercise 7.13 *For some synthetic data estimate the number of local minima for k-means by using the birthday estimate. Is your estimate an unbaised estimate of the number? an upper bound? a lower bound? Why?*

Exercise 7.14 *Examine the example in Figure 7.9 and discuss how to fix it. Optimizing according to the k-center or k-median criteria would seem to produce clustering B while clustering A seems more desirable.*

Exercise 7.15 *Prove that for any two vectors $\mathbf{a}$ and $\mathbf{b}$, $|\mathbf{a} - \mathbf{b}|^2 \geq \frac{1}{2}|\mathbf{a}|^2 - |\mathbf{b}|^2$.*

Exercise 7.16 *Let A be an n×d data matrix, B its best rank k approximation, and C the optimal centers for k-means clustering of rows of A. How is it possible that $\|A - B\|_F^2 < \|A - C\|_F^2$?*

Exercise 7.17 *Suppose S is a finite set of points in space with centroid $\mu(S)$. If a set T of points is added to S, show that the centroid $\mu(S \cup T)$ of $S \cup T$ is at distance at most $\frac{|T|}{|S|+|T|}|\mu(T) - \mu(S)|$ from $\mu(S)$.*

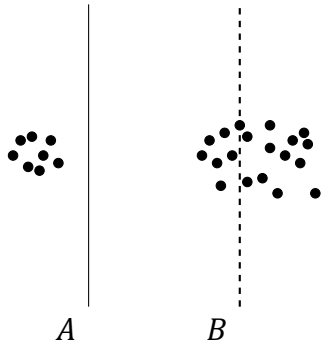


Figure 7.9: insert caption

Exercise 7.18 *What happens if we relax this restriction, for example, if we allow for S, the entire set?*

Exercise 7.19 *Given the graph $G = (V,E)$ of a social network where vertices represent individuals and edges represent relationships of some kind, one would like to define the concept of a community. A number of different definitions are possible.*

1. *A subgraph $S = (V_S,E_S)$ whose density $\frac{E_S}{V_S^2}$ is greater than that of the graph $\frac{E}{V^2}$.*
2. *A subgraph S with a low conductance like property such as the number of graph edges leaving the subgraph normalized by the minimum size of S or $V - S$ where size is measured by the sum of degrees of vertices in S or in $V - S$.*
3. *A subgraph that has more internal edges than in a random graph with the same degree distribution.*

Which would you use and why?

Exercise 7.20 *A stochastic matrix is a matrix with non negative entries in which each row sums to one. Show that for a stochastic matrix, the largest eigenvalue is one. Show that the eigenvalue has multiplicity one if and only if the corresponding Markov Chain is connected.*

Exercise 7.21 *Show that if P is a stochastic matrix and π satisfies $\pi_i p_{ij} = \pi_j p_{ji}$, then for any left eigenvector* **v** *of P, the vector* **u** *with components $u_i = \frac{v_i}{\pi_i}$ is a right eigenvector with the same eigenvalue.*

Exercise 7.22 *Give an example of a clustering problem where the clusters are not linearly separable in the original space, but are separable in a higher dimensional space. Hint: Look at the example for Gaussian kernels in the chapter on learning.*

Exercise 7.23 *The Gaussian kernel maps points to a higher dimensional space. What is this mapping?*

Exercise 7.24 *Agglomerative clustering requires that one calculate the distances between all pairs of points. If the number of points is a million or more, then this is impractical. One might try speeding up the agglomerative clustering algorithm by maintaining a 100 clusters at each unit of time. Start by randomly selecting a hundred points and place each point in a cluster by itself. Each time a pair of clusters is merged randomly select one of the remaining data points and create a new cluster containing that point. Suggest some other alternatives.*

Exercise 7.25 *Let A be the adjacency matrix of an undirected graph. Let* $d(S,S) = \frac{A(S,S)}{|S|}$ *be the density of the subgraph induced by the set of vertices S. Prove that* $d(S,S)$ *is the average degree of a vertex in S. Recall that* $A(S,T) = \sum_{i \in S, j \in T} a_{ij}$

Exercise 7.26 *Suppose A is a matrix with non negative entries. Show that* $A(S,T)/(|S||T|)$ *is maximized by the single edge with highest* a_{ij}*. Recall that* $A(S,T) = \sum_{i \in S, j \in T} a_{ij}$

Exercise 7.27 *Suppose A is a matrix with non negative entries and*

$$\sigma_1(A) = \mathbf{x}^T A \mathbf{y} = \sum_{i,j} x_i a_{ij} y_j, \qquad |\mathbf{x}| = |\mathbf{y}| = 1.$$

Zero out all x_i *less than* $1/2\sqrt{n}$ *and all* y_j *less than* $1/2\sqrt{d}$*. Show that the loss is no more than* $1/4^{th}$ *of* $\sigma_1(A)$*.*

Exercise 7.28 *Consider other measures of density such as* $\frac{A(S,T)}{|S|^\rho |T|^\rho}$ *for different values of* ρ*. Discuss the significance of the densest subgraph according to these measures.*

Exercise 7.29 *Let A be the adjacency matrix of an undirected graph. Let M be the matrix whose* ij^{th} *element is* $a_{ij} - \frac{d_i d_j}{2m}$*. Partition the vertices into two groups S and* $\bar{S}$*. Let s be the indicator vector for the set S and let* $\bar{s}$ *be the indicator variable for* $\bar{S}$*. Then* $s^T Ms$ *is the number of edges in S above the expected number given the degree distribution and* $s^T M\bar{s}$ *is the number of edges from S to* $\bar{S}$ *above the expected number given the degree distribution. Prove that if* $s^T Ms$ *is positive* $s^T M\bar{s}$ *must be negative.*

Exercise 7.30 *Which of the three axioms, scale invariance, richness, and consistency are satisfied by the following clustering algorithms. 1. k-means*

2. Spectral Clustering.

Exercise 7.31 *(Research Problem): What are good measures of density that are also effectively computable? Is there empirical/theoretical evidence that some are better than others?*

Exercise 7.32 *Create a graph with a small community and start a random walk in the community. Calculate the frequency distribution over the vertices of the graph and normalize the frequency distribution by the stationary probability. Plot the ratio of the normalized frequency for the vertices of the graph. What is the shape of the plot for vertices in the small community?*

Exercise 7.33

1. *Create a random graph with the following two structures imbedded in it. The first structure has three equal size communities with no edges between communities and the second structure has five equal size communities with no edges between communities.*

2. *Apply a clustering algorithm to find the dominate structure. Which structure did you get?*

3. *Weaken the dominant structure by removing a fraction of its edges and see if you can find the hidden structure.*

Exercise 7.34 *Experiment with finding hidden communities.*

Exercise 7.35 *Generate a bloc model with two equal size communities where p and q are the probabilities for the edge density within communities and the edge density between communities. Then generated a GNP model with probability* $(p + q)/2$*. Which of two models most likely generates a community with half of the vertices?*

8 Random Graphs

Large graphs appear in many contexts such as the World Wide Web, the internet, social networks, journal citations, and other places. What is different about the modern study of large graphs from traditional graph theory and graph algorithms is that here one seeks statistical properties of these very large graphs rather than an exact answer to questions on specific graphs. This is akin to the switch physics made in the late 19^{th} century in going from mechanics to statistical mechanics. Just as the physicists did, one formulates abstract models of graphs that are not completely realistic in every situation, but admit a nice mathematical development that can guide what happens in practical situations. Perhaps the most basic model is the $G(n,p)$ model of a random graph. In this chapter, we study properties of the $G(n,p)$ model as well as other models.

8.1 The $G(n,p)$ Model

The $G(n,p)$ model, due to Erdo¨s and R´enyi, has two parameters, n and p. Here n is the number of vertices of the graph and p is the edge probability. For each pair of distinct vertices, v and w, p is the probability that the edge (v,w) is present. The presence of each edge is statistically independent of all other edges. The graph-valued random variable

with these parameters is denoted by $G(n,p)$. When we refer to "the graph $G(n,p)$", we mean one realization of the random variable. In many cases, p will be a function of n such as $p = d/n$ for some constant d. For example, if $p = d/n$ then the expected degree of a vertex of the graph is $(n-1)\frac{d}{n} \approx d$. In order to simplify calculations in this chapter, we will often use the approximation that $\frac{n-1}{n} \approx 1$. In fact, conceptually it is helpful to think of n as both the total number of vertices and as the number of potential neighbors of any given node, even though the latter is really $n - 1$; for all our calculations, when n is large, the correction is just a low-order term.

The interesting thing about the $G(n,p)$ model is that even though edges are chosen independently with no "collusion", certain global properties of the graph emerge from the independent choices. For small p, with $p = d/n$, $d < 1$, each connected component in the graph is small. For $d > 1$, there is a giant component consisting of a constant fraction of the vertices. In addition, there is a rapid transition at the threshold $d = 1$. Below the threshold, the probability of a giant component is very small, and above the threshold, the probability is almost one.

The phase transition at the threshold $d = 1$ from very small $o(n)$ size components to a giant $\Omega(n)$ sized component is illustrated by the following example. Suppose the vertices represent people and an edge means the two people it connects know each other. Given a chain of connections, such as A knows B, B knows C, C knows D, ..., and Y knows Z, we say that A indirectly knows Z. Thus, all people belonging to a connected component of the graph indirectly know each other. Suppose each pair of people, independent of other pairs, tosses a coin that comes up heads with probability $p = d/n$. If it is heads, they

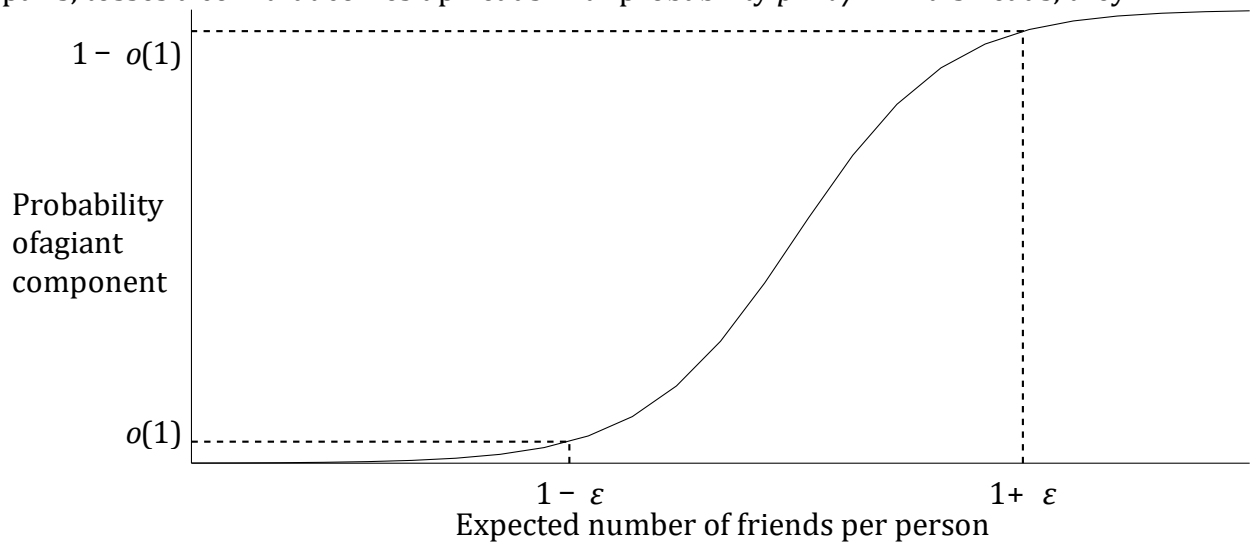


Figure 8.1: Probability of a giant component as a function of the expected number of people each person knows directly.

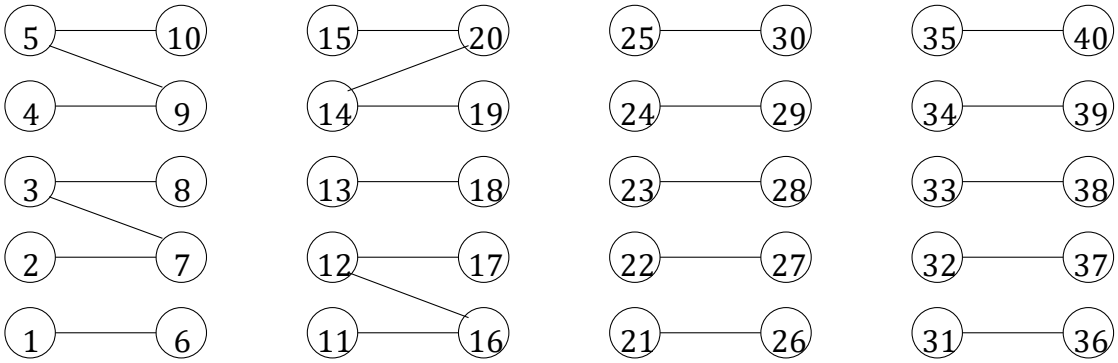
know each other; if it comes up tails, they don't. The value of d can be interpreted as the expected number of people a single person directly knows. The question arises as to how large are sets of people who indirectly know each other?

If the expected number of people each person knows is more than one, then a giant component of people, all of whom indirectly know each other, will be present consisting of a constant fraction of all the people. On the other hand, if in expectation, each person knows less than one person, the largest set of people who know each other indirectly is a vanishingly small fraction of the whole. Furthermore, the transition from the vanishing fraction to a constant fraction of the whole, happens abruptly between d slightly less than one to d slightly more than one. See Figure 8.1. Note that there is no global coordination of who knows whom. Each pair of individuals decides independently. Indeed, many large real-world graphs, with constant average degree, have a giant component. This is perhaps the most important global property of the $G(n,p)$ model.

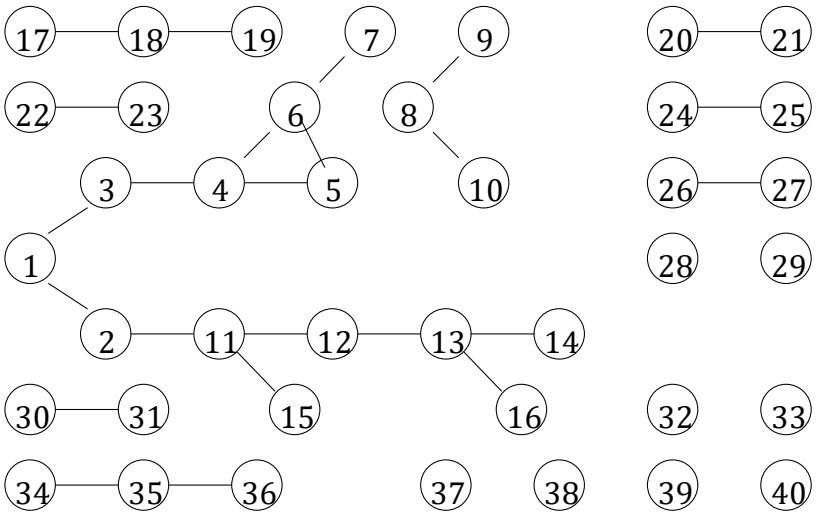
8.1.1 Degree Distribution

One of the simplest quantities to observe in a real graph is the number of vertices of given degree, called the vertex degree distribution. It is also very simple to study these distributions in $G(n,p)$ since the degree of each vertex is the sum of n independent random variables, which results in a binomial distribution.

Example: In $G(n, \frac{1}{2})$, each vertex is of degree close to $n/2$. In fact, for any $\varepsilon > 0$, the degree of each vertex almost surely is within $1 \pm \varepsilon$ times $n/2$. To see this, note that the degree of a vertex is the sum of $n - 1 \approx n$ indicator variables that take on value one or



A graph with 40 vertices and 24 edges



A randomly generated $G(n,p)$ graph with 40 vertices and 24 edges

Figure 8.2: Two graphs, each with 40 vertices and 24 edges. The second graph was randomly generated using the $G(n,p)$ model with $p = 1.2/n$. A graph similar to the top graph is almost surely not going to be randomly generated in the $G(n,p)$ model, whereas a graph similar to the lower graph will almost surely occur. Note that the lower graph consists of a giant component along with a number of small components that are trees.

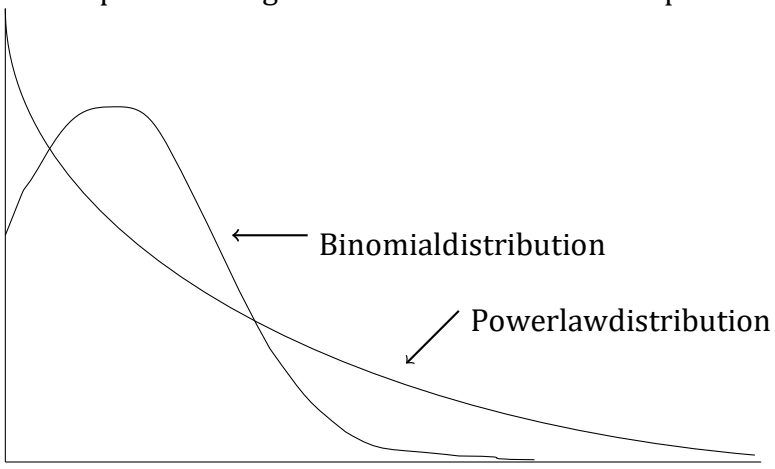


Figure 8.3: Illustration of the binomial and the power law distributions.

zero depending on whether the edge is present or not, each of mean $\frac{1}{2}$ and variance $\frac{1}{4}$. The expected value of the sum is the sum of the expected values and $\approx \frac{n}{2}$ the degree $\approx \frac{n}{4}$ has variance of the sum is the sum of the variances, and hence meanand variance. Thus,

$\sqrt{}$

the probability mass is within an additive term of $\pm c\,n$ of the mean for some constant c and thus within a multiplicative factor of $1^{\pm \epsilon \text{ of } \frac{n}{2}}$ for sufficiently large n. ■

The degree distribution of $G(n,p)$ for general p is also binomial. Since p is the probability of an edge being present, the expected degree of a vertex is $p(n-1) \approx pn$. The degree distribution is given by

$$\text{Prob(vertex has degree } k) = \binom{n-1}{k} p^k (1-p)^{n-k-1} \approx \binom{n}{k} p^k (1-p)^{n-k}.$$

The quantity $\binom{n}{k}$ is the number of ways of choosing k edges, out of the possible n edges, and $p^k(1-p)^{n-k}$ is the probability that the k selected edges are present and the remaining $n-k$ are not.

The binomial distribution falls off exponentially fast as one moves away from the mean. However, the degree distributions of graphs that appear in many applications do not exhibit such sharp drops. Rather, the degree distributions are much broader. This is often referred to as having a "heavy tail". The term tail refers to values of a random variable far away from its mean, usually measured in number of standard deviations. Thus, although the $G(n,p)$ model is important mathematically, more complex models are needed to represent real world graphs.

Consider an airline route graph. The graph has a wide range of degrees from degree one or two for a small city to degree 100 or more, for a major hub. The degree distribution is not binomial. Many large graphs that arise in various applications appear to have power law degree distributions. A power law degree distribution is one in which the number of vertices having a given degree decreases as a power of the degree, as in

$$\text{Number(degree } k \text{ vertices)} = \frac{c\,n}{k^r},$$

for some small positive real r, often just slightly less than three. Later, we will consider a random graph model giving rise to such degree distributions.

The following theorem states that the degree distribution of the random graph $G(n,p)$ is tightly concentrated about its expected value. That is, the probability that the degree of a vertex differs from its expected degree by more than $\lambda \sqrt{np}$, drops off exponentially fast with λ.

Theorem 8.1 *Let v be a vertex of the random graph $G(n,p)$. Let α be a real number in $(0, \sqrt{np})$.*

$$Prob(|np - deg(v)| \geq \alpha \sqrt{np}) \leq 3e^{-\alpha^2/8}.$$

Proof: The degree $\deg(v)$ is the sum of $n-1$ independent Bernoulli random variables, $x_1, x_2, \ldots, x_{n-1}$, where, x_i is the indicator variable that the i^{th} edge from v is present. So, approximating $n-1$ with n, the theorem follows from Theorem 12.6 in the appendix.

■

Although the probability that the degree of a single vertex differs significantly from its expected value drops exponentially, the statement that the degree of every vertex is close to its expected value requires that p is $\Omega(\frac{\log n}{n})$. That is, the expected degree grows at least logarithmically with the number of vertices.

Corollary 8.2 *Suppose ε is a positive constant. If* $p \geq \frac{9 \ln n}{n\varepsilon^2}$, *then almost surely every vertex has degree in the range* $(1 - \varepsilon)np$ *to* $(1 + \varepsilon)np$.

Proof: Apply Theorem 8.1 with $\alpha = \varepsilon \sqrt{np}$ to get that the probability that an individual vertex has degree outside the range $[(1 - \varepsilon)np, (1 + \varepsilon)np]$ is at most $3e^{-\varepsilon^2 np/8}$. By the union bound, the probability that some vertex has degree outside this range is at most $3ne^{-\varepsilon^2 np/8}$. For this to be $o(1)$, it suffices for $p \geq \frac{9 \ln n}{n\varepsilon^2}$. ■

Note that the assumption p is $\Omega(\frac{\log n}{n})$ is necessary. If $p = d/n$ for d a constant, then some vertices may well have degrees outside the range $[(1 - \varepsilon)d, (1 + \varepsilon)d]$. Indeed, shortly we will see that it is highly likely that for $p = \frac{1}{n}$ there is a vertex of degree $\Omega(\log n / \log\log n)$. Moreover, for $p = \frac{1}{n}$ it is easy to see that with high probability there will be at least one vertex of degree zero.

When p is a constant, the expected degree of vertices in $G(n,p)$ increases with n. In $G\left(n, \frac{1}{2}\right)$ the expected degree of a vertex is approximately $n/2$. In many real applications, we will be concerned with $G(n,p)$ where $p = d/n$, for d a constant, i.e., graphs whose expected degree is a constant d independent of n. As n goes to infinity, the binomial distribution with $p = \frac{d}{n}$

$$\text{Prob}(k) = \binom{n}{k} \left(\frac{d}{n}\right)^k \left(1 - \frac{d}{n}\right)^{n-k}$$

approaches the Poisson distribution

$$\text{Prob}(k) = \frac{d^k}{k!} e^{-d}$$

To see this, assume $k = o(n)$ and use the approximations $\binom{n}{k} \approx \frac{n^k}{k!}$, $n - k \approx n$, and $\left(1 - \frac{d}{n}\right)^{n-k} \approx \left(1 - \frac{d}{n}\right)^n \approx e^{-d}$. Then

$$\lim_{n\to\infty} \binom{n}{k} \left(\frac{d}{n}\right)^k \left(1 - \frac{d}{n}\right)^{n-k} = \frac{n^k}{k!} \frac{d^k}{n^k} e^{-d} = \frac{d^k}{k!} e^{-d}.$$

Note that for $p = \frac{d}{n}$, where d is a constant independent of n, the probability of the binomial distribution falls off rapidly for $k > d$, and is essentially zero once $k!$ dominates d^k. This justifies the $k = o(n)$ assumption. Thus, the Poisson distribution is a good approximation.

Example: In $G(n, \frac{1}{n})$ many vertices are of degree one, but not all. Some are of degree zero and some are of degree greater than one. In fact, it is highly likely that there is a vertex of degree $\Omega(\log n/\log\log n)$. The probability that a given vertex is of degree k is

$$\text{Prob}(k) = \binom{n-1}{k}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-1-k} \approx \binom{n}{k}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-k} \approx \frac{e^{-1}}{k!}.$$

If $k = \log n/\log\log n$,

$$\log k^k = k\log k = \frac{\log n}{\log\log n}(\log\log n - \log\log\log n) \leq \log n$$

and thus $k^k \leq n$. Since $k! \leq k^k \leq n$, the probability that a vertex has degree $k = \log n/\log\log n$ is at least $\frac{1}{k!}e^{-1} \geq \frac{1}{en}$. If the degrees of vertices were independent random variables, then this would be enough to argue that there would be a vertex of degree $\log n/\log\log n$ with probability at least $1 - \left(1-\frac{1}{en}\right)^n = 1 - e^{-\frac{1}{e}} \cong 0.31$. But the degrees are not quite independent since when an edge is added to the graph it affects the degree of two vertices. This is a minor technical point, which one can get around. ■

8.1.2 Existence of Triangles in $G(n,d/n)$

What is the expected number of triangles in $G\left(n, \frac{d}{n}\right)$, when d is a constant? As the number of vertices increases one might expect the number of triangles to increase, but this is not the case. Although the number of triples of vertices grows as n^3, the probability of an edge between two specific vertices decreases linearly with n. Thus, the probability of all three edges between the pairs of vertices in a triple of vertices being present goes down as n^{-3}, exactly canceling the rate of growth of triples.

A random graph with n vertices and edge probability d/n, has an expected number of triangles that is independent of n, namely $d^3/6$. There are $\binom{n}{3}$ triples of vertices. Each triple has probability $\left(\frac{d}{n}\right)^3$ of being a triangle. Let Δ_{ijk} be the indicator variable for the triangle with vertices i, j, and k being present. That is, all three edges (i,j), (j,k), and (i,k) being present. Then the number of triangles is $x = \sum_{ijk}\Delta_{ijk}$. Even though the existence of the triangles are not statistically independent events, by linearity of expectation, which does not assume independence of the variables, the expected value of a sum of random variables is the sum of the expected values. Thus, the expected number of triangles is

$$E(x) = E\left(\sum_{ijk}\Delta_{ijk}\right) = \sum_{ijk}E(\Delta_{ijk}) = \binom{n}{3}\left(\frac{d}{n}\right)^3 \approx \frac{d^3}{6}.$$

Even though on average there are $\frac{d^3}{6}$ triangles per graph, this does not mean that with high probability a graph has a triangle. Maybe half of the graphs have $\frac{d^3}{3}$ triangles and the other half have none for an average of $\frac{d^3}{6}$ triangles. Then, with probability 1/2, a graph

selected at random would have no triangle. If $1/n$ of the graphs had $\frac{d^3}{6}n$ triangles and the remaining graphs had no triangles, then as n goes to infinity, the probability that a graph selected at random would have a triangle would go to zero.

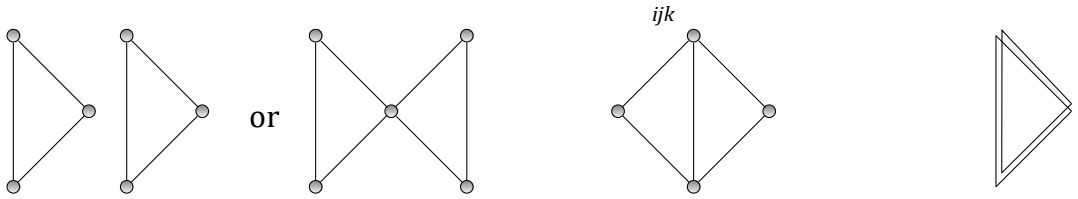
We wish to assert that with some nonzero probability there is at least one triangle in $G(n,p)$ when $p = \frac{d}{n}$. If all the triangles were on a small number of graphs, then the number of triangles in those graphs would far exceed the expected value and hence the variance would be high. A second moment argument rules out this scenario where a small fraction of graphs have a large number of triangles and the remaining graphs have none.

Let's calculate $E(x^2)$ where x is the number of triangles. Write x as $x = \sum_{ijk} \Delta_{ijk}$, where Δ_{ijk} is the indicator variable of the triangle with vertices $i,j,$ and k being present. Expanding the squared term

$$E(x^2) = E\left(\sum_{i,j,k} \Delta_{ijk}\right)^2 = E\left(\sum_{\substack{i,\,j,\,k\\ i',j',k'}} \Delta_{ijk}\Delta_{i'j'k'}\right).$$

Split the above sum into three parts. In Part 1, let S_1 be the set of i,j,k and i^0,j^0,k^0 which share at most one vertex and hence the two triangles share no edge. In this case, Δ_{ijk} and $\Delta_{i^0j^0k^0}$ are independent and

$$E\left(\sum_{S_1} \Delta_{ijk}\Delta_{i'j'k'}\right) = \sum_{S_1} E(\Delta_{ijk})E(\Delta_{i'j'k'}) \le \left(\sum_{\text{all}} E(\Delta_{ijk})\right)\left(\sum_{\substack{\text{all}\\ i'j'k'}} E(\Delta_{i'j'k'})\right) = E^2(x).$$



The two triangles of Part 1 are either disjoint or share at most one vertex

The two triangles in Part 2 share an edge

The two triangles of Part 3 are the same tri-angle

Figure 8.4: The triangles in Part 1, Part 2, and Part 3 of the second moment argument for the existence of triangles in $G(n, \frac{d}{n})$.

In Part 2, i,j,k and i^0,j^0,k^0 share two vertices and hence one edge. See Figure 8.4. Four vertices and five edges are involved overall. There are at most $\binom{n}{4} \in O(n^4)$, 4-vertex subsets and $\binom{4}{2}$ ways to partition the four vertices into two triangles with a common edge. The probability of all five edges in the two triangles being present is p^5, so this part sums to $O(n^4p^5) = O(d^5/n)$ and is $o(1)$. There are so few triangles in the graph, the probability of two triangles sharing an edge is extremely unlikely.

In Part 3, i,j,k and i^0,j^0,k^0 are the same sets. The contribution of this part of the summation to $E(x^2)$ is $\binom{n}{3}p^3 = \frac{d^3}{6}$. Thus, putting all three parts together, we have:

$$E(x^2) \leq E^2(x) + \frac{d^3}{6} + o(1),$$

which implies

$$\mathrm{Var}(x) = E(x^2) - E^2(x) \leq \frac{d^3}{6} + o(1).$$

For x to be equal to zero, it must differ from its expected value by at least its expected value. Thus,

$$\text{Prob}(x = 0) \leq \text{Prob}\big(|x - E(x)| \geq E(x)\big).$$

By Chebychev inequality,

$$\text{Prob}(x = 0) \leq \frac{\mathrm{Var}(x)}{E^2(x)} \leq \frac{d^3/6 + o(1)}{d^6/36} \leq \frac{6}{d^3} + o(1). \tag{8.1}$$

Thus, for $d > \sqrt[3]{6} \approx 1.8$, Prob($x$ = 0) < 1 and $G(n,p)$ has a triangle with nonzero probability. For $d < \sqrt[3]{6},\ E(x) = \frac{d^3}{6} < 1$ and there simply are not enough edges in the graph for there to be a triangle.

8.2 Phase Transitions

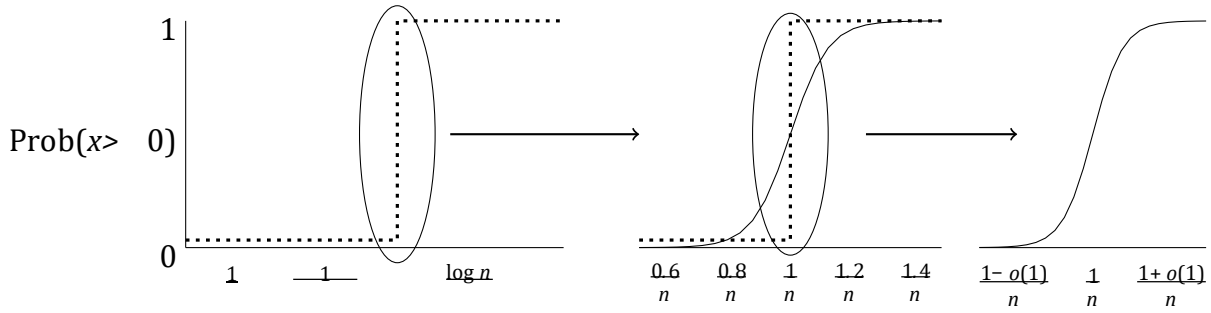
Many properties of random graphs undergo structural changes as the edge probability passes some threshold value. This phenomenon is similar to the abrupt phase transitions in physics, as the temperature or pressure increases. Some examples of this are the abrupt appearance of cycles in $G(n,p)$ when p reaches $1/n$ and the disappearance of isolated vertices when p reaches $\frac{\ln n}{n}$. The most important of these transitions is the emergence of a giant component, a connected component of size $\Theta(n)$, which happens at $d = 1$. Recall Figure 8.1.

Probability	Transition
$p = o(\frac{1}{n})$	Forest of trees, no component of size greater than $O(\log n)$
$p = \frac{d}{n}, d < 1$	Cycles appear, no component of size greater than $O(\log n)$
$p = \frac{d}{n}, d = 1$	Components of size $O(n^{\frac{2}{3}})$
$p = \frac{d}{n}, d > 1$	Giant component plus $O(\log n)$ components

	vertices
	Diameter two ent plus isolated
$p = \frac{\ln n}{n}$	Disappearance of isolated vertices Appearance of Hamilton circuit Diameter $O(\log n)$
$p = \frac{1}{2}$	Clique of size $(2 - \epsilon) \ln n$

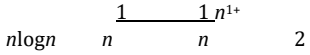
Table 1: Phase transitions

For these and many other properties of random graphs, a threshold exists where an abrupt transition from not having the property to having the property occurs. If there exists a function $p(n)$ such that when $\lim_{n\to\infty} \frac{p_1(n)}{p(n)} = 0$, $G(n, p_1(n))$ almost surely does not have the property, and when $\lim_{n\to\infty} \frac{p_2(n)}{p(n)} = \infty$, $G(n, p_2(n))$ almost surely has the property, then we say that a *phase transition* occurs, and $p(n)$ is the *threshold*. Recall that $G(n,p)$ "almost surely does not have the property" means that the probability that it has the property goes to zero in the limit, as n goes to infinity. We shall soon see that every increasing property has a threshold. This is true not only for increasing properties of $G(n,p)$, but for increasing properties of any combinatorial structure. If for $cp(n)$, $c < 1$, the



graph almost surely does not have the property and for $cp(n)$, $c > 1$, the graph almost surely has the property, then $p(n)$ is a *sharp threshold*. The existence of a giant component has a sharp threshold at $1/n$. We will prove this later.

In establishing phase transitions, we often use a variable $x(n)$ to denote the number of occurrences of an item in a random graph. If the expected value of $x(n)$ goes to zero as n goes to infinity, then a graph picked at random almost surely has no occurrence of the



(*a*)

Figure 8.5: Figure 8.5(a) shows a phase transition at $p = \frac{1}{n}$. The dotted line shows an abrupt transition in Prob(x) from 0 to 1. For any function asymptotically less than $\frac{1}{n}$, Prob(x)>0 is zero and for any function asymptotically greater than $\frac{1}{n}$, Prob(x)>0 is one. Figure 8.5(b) expands the scale and shows a less abrupt change in probability unless the phase transition is sharp as illustrated by the dotted line. Figure 8.5(c) is a further expansion and the sharp transition is now more smooth.

item. This follows from Markov's inequality. Since x is a nonnegative random variable Prob($x \geq a) \leq \frac{1}{a}E(x$), which implies that the probability of $x(n) \geq 1$ is at most $E(x(n))$. That is, if the expected number of occurrences of an item in a graph goes to zero, the probability that there are one or more occurrences of the item in a randomly selected graph goes to zero. This is called the *first moment method*.

The previous section showed that the property of having a triangle has a threshold at $p(n) = 1/n$. If the edge probability $p_1(n)$ is $o(1/n)$, then the expected number of triangles goes to zero and by the first moment method, the graph almost surely has no triangle. However, if the edge probability $p_2(n)$ satisfies $\frac{p_2(n)}{1/n} \to \infty$, then from (8.1), the probability of having no triangle is at most $6/d^3 + o(1) = 6/(np_2(n))^3 + o(1)$, which goes to zero. This latter case uses what we call the second moment method. The first and second moment methods are broadly used. We describe the second moment method in some generality now.

When the expected value of $x(n)$, the number of occurrences of an item, goes to infinity, we cannot conclude that a graph picked at random will likely have a copy since the items may all appear on a vanishingly small fraction of the graphs. We resort to a technique called the *second moment method*. It is a simple idea based on Chebyshev's inequality.

Theorem 8.3 (Second Moment method) *Let $x(n)$ be a random variable with $E(x) > 0$. If*

$$Var(x) = o\left(E^2(x)\right),$$

then x is almost surely greater than zero.

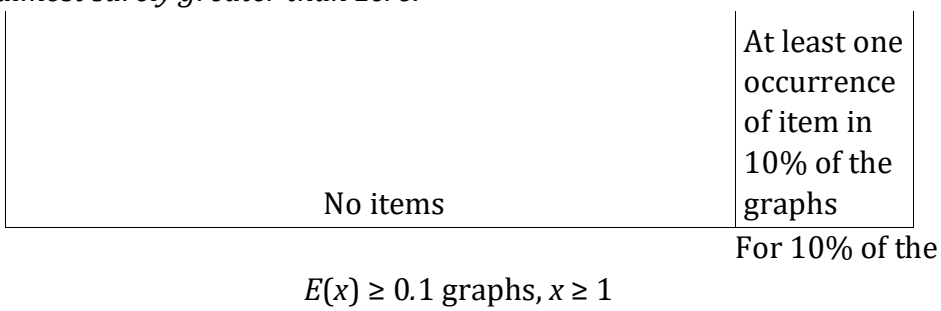


Figure 8.6: If the expected fraction of the number of graphs in which an item occurs did not go to zero, then $E(x)$, the expected number of items per graph, could not be zero. Suppose 10% of the graphs had at least one occurrence of the item. Then the expected number of occurrences per graph must be at least 0.1. Thus, $E(x) \rightarrow 0$ implies the probability that a graph has an occurrence of the item goes to zero. However, the other direction needs more work. If $E(x)$ is large, a second moment argument is needed to conclude that the probability that a graph picked at random has an occurrence of the item is nonnegligible, since there could be a large number of occurrences concentrated on a vanishingly small fraction of all graphs. The second moment argument claims that for a nonnegative random variable x with $E(x) > 0$, if $\text{Var}(x)$ is $o(E^2(x))$ or alternatively if $E(x^2) \leq E^2(x)(1 + o(1))$, then almost surely $x > 0$.

Proof: If $E(x) > 0$, then for x to be less than or equal to zero, it must differ from its expected value by at least its expected value. Thus,

$$\text{Prob}(x \leq 0) \leq \text{Prob}\Big(|x - E(x)| \geq E(x)\Big).$$

By Chebyshev inequality

$$\text{Prob}\Big(|x - E(x)| \geq E(x)\Big) \leq \frac{\text{Var}(x)}{E^2(x)} \rightarrow 0.$$

Thus, $\text{Prob}(x \leq 0)$ goes to zero if $\text{Var}(x)$ is $o(E^2(x))$. ∎

Corollary 8.4 *Let x be a random variable with $E(x) > 0$. If*

$$E(x^2) \leq E^2(x)\big(1 + o(1)\big),$$

then x is almost surely greater than zero.

Proof: If $E(x^2) \leq E^2(x)(1 + o(1))$, then

$$Var(x) = E(x^2) - E^2(x) \leq E^2(x)o(1) = o\big(E^2(x)\big).$$

∎

Second moment arguments are more difficult than first moment arguments since they deal with variance and without independence we do not have $E(xy) = E(x)E(y)$. In the triangle example, dependence occurs when two triangles share a common edge. However, if $p = \frac{d}{n}$, there are so few triangles that almost surely no two triangles share a common edge and the lack of statistical independence does not affect the answer. In looking for a phase transition, almost always the transition in probability of an item being present occurs when the expected number of items transitions.

Threshold for graph diameter two (two degrees of separation)

We now present the first example of a sharp phase transition for a property. This means that slightly increasing the edge probability p near the threshold takes us from almost surely not having the property to almost surely having it. The property is that of a random graph having diameter less than or equal to two. The diameter of a graph is the maximum length of the shortest path between a pair of nodes. In other words, the property is that every pair of nodes has "at most two degrees of separation".

The following technique for deriving the threshold for a graph having diameter two is a standard method often used to determine the threshold for many other objects. Let x be a random variable for the number of objects such as triangles, isolated vertices, or Hamiltonian circuits, for which we wish to determine a threshold. Then we determine the value of p, say p_0, where the expected value of x goes from vanishingly small to unboundedly large. For $p < p_0$ almost surely a graph selected at random will not have a copy of the item. For $p > p_0$, a second moment argument is needed to establish that the items are not concentrated on a vanishingly small fraction of the graphs and that a graph picked at random will almost surely have a copy.

Our first task is to figure out what to count to determine the threshold for a graph having diameter two. A graph has diameter two if and only if for each pair of vertices i and j, either there is an edge between them or there is another vertex k to which both i and j have an edge. So, what we will count is the number of pairs i and j that fail, i.e., the number of pairs i and j that have more than two degrees of separation. The set of neighbors of i and the set of neighbors of j are random subsets of expected cardinality np. For these two sets to intersect requires $np \approx \sqrt{n}$ or $p \approx \frac{1}{\sqrt{n}}$. Such statements often go under the general name of "birthday paradox" though it is not a paradox. In what follows, we will prove a threshold of $O(\sqrt{\ln n / n})$ for a graph to have diameter two. The extra factor of $\sqrt{\ln n}$ ensures that every one of the $\binom{n}{2}$ pairs of i and j has a common neighbor. $p = c\sqrt{\frac{\ln n}{n}}$ When, for $c < \sqrt{2}$, the graph almost surely has diameter greater than two and for $c > \sqrt{2}$, the graph almost surely has diameter less than or equal to two.

Theorem 8.5 *The property that $G(n,p)$ has diameter two has a sharp threshold at* $p = \sqrt{2}\sqrt{\frac{\ln n}{n}}$.

Proof: If G has diameter greater than two, then there exists a pair of nonadjacent vertices i and j such that no other vertex of G is adjacent to both i and j. This motivates calling such a pair *bad*.

Introduce a set of indicator variables I_{ij}, one for each pair of vertices (i,j) with $i < j$, where I_{ij} is 1 if and only if the pair (i,j) is bad. Let

$$x = \sum_{i<j} I_{ij}$$

be the number of bad pairs of vertices. Putting $i < j$ in the sum ensures each pair (i,j) is counted only once. A graph has diameter at most two if and only if it has no bad pair, i.e., $x = 0$. Thus, if $\lim_{n\to\infty} E(x) = 0$, then for large n, almost surely, a graph has no bad pair and hence has diameter at most two.

The probability that a given vertex is adjacent to both vertices in a pair of vertices (i,j) is p^2. Hence, the probability that the vertex is not adjacent to both vertices is $1 - p^2$. The probability that no vertex is adjacent to the pair (i,j) is $(1 - p^2)^{n-2}$ and the probability that i and j are not adjacent is $1 - p$. Since there are $\binom{n}{2}$ pairs of vertices, the expected number of bad pairs is

$$E(x) = \binom{n}{2}(1-p)\left(1-p^2\right)^{n-2}.$$

Setting $p = c\sqrt{\frac{\ln n}{n}}$,

$$\begin{aligned} E(x) &\cong \frac{n^2}{2}\left(1 - c\sqrt{\tfrac{\ln n}{n}}\right)\left(1 - c^2\tfrac{\ln n}{n}\right)^n \\ &\cong \tfrac{n^2}{2}e^{-c^2\ln n} \\ &\cong \tfrac{1}{2}n^{2-c^2}. \end{aligned}$$

For $c > \sqrt{2}$, $\lim_{n\to\infty} E(x) = 0$. By the first moment method, $p = c\sqrt{\frac{\ln n}{n}}$ forwith $c > \sqrt{2}$, $G(n,p)$ almost surely has no bad pair and hence has diameter at most two.

Next, consider the case $c < \sqrt{2}$ where $\lim_{n\to\infty} E(x) = \infty$. We appeal to a second moment argument to claim that almost surely a graph has a bad pair and thus has diameter greater than two.

$$E(x^2) = E\left(\sum_{i<j} I_{ij}\right)^2 = E\left(\sum_{i<j} I_{ij}\sum_{k<l} I_{kl}\right) = E\left(\sum_{\substack{i<j\\k<l}} I_{ij}I_{kl}\right) = \sum_{\substack{i<j\\k<l}} E(I_{ij}I_{kl}).$$

The summation can be partitioned into three summations depending on the number of distinct indices among i,j,k, and l. Call this number a.

$$E\left(x^2\right) = \sum_{\substack{i<j \\ k<l}} E\left(I_{ij}I_{kl}\right) + \sum_{\substack{\{i,j,k\} \\ i<j}} E\left(I_{ij}I_{ik}\right) + \sum_{i<j} E\left(I_{ij}^2\right) \quad . \tag{8.2}$$

$a = 4$ $a = 3$ $a = 2$

Consider the case $a = 4$ where *i,j,k,* and *l* are all distinct. If $I_{ij}I_{kl} = 1$, then both pairs (*i,j*) and (*k,l*) are bad and so for each *u* not in {*i,j,k,l*}, at least one of the edges (*i,u*) or (*j,u*) is absent and, in addition, at least one of the edges (*k,u*) or (*l,u*) is absent. The probability of this for one *u* not in {*i,j,k,l*} is $(1 - p^2)^2$. As *u* ranges over all the $n - 4$ vertices not in {*i,j,k,l*}, these events are all independent. Thus,

$$E(I_{ij}I_{kl}) \le \left(1-p^2\right)^{2(n-4)} \le \left(1 - c^2\frac{\ln n}{n}\right)^{2n}\left(1+o(1)\right) \le n^{-2c^2}\left(1+o(1)\right)$$

and the first sum is

$$\sum_{\substack{i<j \\ k<l}} E(I_{ij}I_{kl}) \le \frac{1}{4}n^{4-2c^2}\left(1+o(1)\right) \quad ,$$

where, the $\frac{1}{4}$ is because only a fourth of the 4-tupples (*i,j,k,l*) have $i < j$ and $k < l$.

For the second summation, observe that if $I_{ij}I_{ik} = 1$, then for every vertex *u* not equal to *i*, *j,* or *k*, either there is no edge between *i* and *u* or there is an edge (*i,u*) and both edges (*j,u*) and (*k,u*) are absent. The probability of this event for one *u* is

$$1 - p + p(1-p)^2 = 1 - 2p^2 + p^3 \approx 1 - 2p^2.$$

Thus, the probability for all such *u* is $(1 - 2p^2)^{n-3}$. Substituting $\overline{\qquad}\ c\sqrt{\frac{\ln n}{n}}$ for p yields

$$\left(1 - \frac{2c^2\ln n}{n}\right)^{n-3} \cong e^{-2c^2\ln n} = n^{-2c^2} \quad ,$$

which is an upper bound on $E(I_{ij}I_{kl})$ for one *i,j,k,* and *l* with $a = 3$. Summing over all distinct triples yields n^{3-2c_2} for the second summation in (8.2).

For the third summation, since the value of I_{ij} is zero or one, $E\left(I_{ij}^2\right) = E\left(I_{ij}\right)$. Thus,

$$\sum_{ij} E\left(I_{ij}^2\right) = E\left(x\right) \quad .$$

$E\left(x^2\right) \le \frac{1}{4}n^{4-2c^2} + n^{3-2c^2} + n^{2-c^2}$ $E\left(x\right) \cong \frac{1}{2}n^{2-c^2}$ Hence,and, from which it follows that

√

for $c < 2$, $E(x^2) \leq E^2(x)(1 + o(1))$. By a second moment argument, Corollary 8.4, a graph almost surely has at least one bad pair of vertices and thus has diameter greater than two. Therefore, the property that the diameter of $G(n,p)$ is less than or equal to two has a sharp threshold at $p = \sqrt{2}\sqrt{\frac{\ln n}{n}}$ ■

Disappearance of Isolated Vertices

The disappearance of isolated vertices in $G(n,p)$ has a sharp threshold at $\frac{\ln n}{n}$. At this point the giant component has absorbed all the small components and with the disappearance of isolated vertices, the graph becomes connected.

Theorem 8.6 *The disappearance of isolated vertices in $G(n,p)$ has a sharp threshold of $\frac{\ln n}{n}$.*

Proof: Let x be the number of isolated vertices in $G(n,p)$. Then,

$$E(x) = n(1-p)^{n-1}.$$

Since we believe the threshold to be $\frac{\ln n}{n}$, consider $p = c\frac{\ln n}{n}$. Then,

$$\lim_{n\to\infty} E(x) = \lim_{n\to\infty} n\left(1 - \frac{c\ln n}{n}\right)^n = \lim_{n\to\infty} ne^{-c\ln n} = \lim_{n\to\infty} n^{1-c}.$$

If $c > 1$, the expected number of isolated vertices, goes to zero. If $c < 1$, the expected number of isolated vertices goes to infinity. If the expected number of isolated vertices goes to zero, it follows that almost all graphs have no isolated vertices. On the other hand, if the expected number of isolated vertices goes to infinity, a second moment argument is needed to show that almost all graphs have an isolated vertex and that the isolated vertices are not concentrated on some vanishingly small set of graphs with almost all graphs not having isolated vertices.

Assume $c < 1$. Write $x = I_1 + I_2 + \cdots + I_n$ where I_i is the indicator variable indicating whether vertex i is an isolated vertex. Then $E(x^2) = \sum_{i=1}^{n} E(I_i^2) + 2\sum_{i<j} E(I_iI_j)$. Since I_i equals 0 or 1, $I_i^2 = I_i$ and the first sum has value $E(x)$. Since all elements in the second sum are equal

$$\begin{aligned} E(x^2) &= E(x) + n(n-1)E(I_1I_2) \\ &= E(x) + n(n-1)(1-p)^{2(n-1)-1}. \end{aligned}$$

The minus one in the exponent $2(n-1)-1$ avoids counting the edge from vertex 1 to vertex 2 twice. Now,

$$\begin{aligned} \frac{E(x^2)}{E^2(x)} &= \frac{n(1-p)^{n-1} + n(n-1)(1-p)^{2(n-1)-1}}{n^2(1-p)^{2(n-1)}} \\ &= \frac{1}{n(1-p)^{n-1}} + \left(1 - \frac{1}{n}\right)\frac{1}{1-p}. \end{aligned}$$

For $p = c\frac{\ln n}{n}$ with $c < 1$, $\lim_{n\to\infty} E(x) = \infty$ and

$$\lim_{n\to\infty} \frac{E\left(x^2\right)}{E^2\left(x\right)} = \lim_{n\to\infty} \left[\frac{1}{n^{1-c}} + \left(1-\frac{1}{n}\right)\frac{1}{1-c\frac{\ln n}{n}}\right] = \lim_{n\to\infty}\left(1+c\frac{\ln n}{n}\right) = o(1)+1$$

259

.

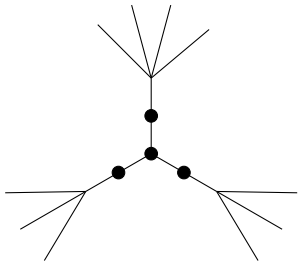


Figure 8.7: A degree three vertex with three adjacent degree two vertices. Graph cannot have a Hamilton circuit.

By the second moment argument, Corollary 8.4, the probability that $x = 0$ goes to zero implying that almost all graphs have an isolated vertex. Thus, $\frac{\ln n}{n}$ is a sharp threshold for the disappearance of isolated vertices. For $p = c\frac{\ln n}{n}$, when $c > 1$ there almost surely are no isolated vertices, and when $c < 1$ there almost surely are isolated vertices. ■

Hamilton circuits

So far in establishing phase transitions in the $G(n,p)$ model for an item such as the disappearance of isolated vertices, we introduced a random variable x that was the number of occurrences of the item. We then determined the probability p for which the expected value of x went from zero to infinity. For values of p for which $E(x) \to 0$, we argued that with high probability, a graph generated at random had no occurrences of x. For values of x for which $E(x) \to \infty$, we used the second moment argument to conclude that with high probability, a graph generated at random had occurrences of x. That is, the occurrences that forced $E(x)$ to infinity were not all concentrated on a vanishingly small fraction of the graphs. One might raise the question for the $G(n,p)$ graph model, do there exist items that are so concentrated on a small fraction of the graphs that the value of p where $E(x)$ goes from zero to infinity is not the threshold? An example where this happens is Hamilton circuits.

A Hamilton circuit is a simple cycle that includes all the vertices. For example, in a graph of 4 vertices, there are three possible Hamilton circuits: (1,2,3,4), (1,2,4,3), and (1,3,2,4). Note that our graphs are undirected, so the circuit (1,2,3,4) is the same as the circuit (1,4,3,2).

Let *x* be the number of Hamilton circuits in *G*(*n,p*) and let $p = \frac{d}{n}$ for some constant *d.* There are $\frac{1}{2}(n-1)!$ potential Hamilton circuits in a graph and each has probability $(\frac{d}{n})^n$ of actually being a Hamilton circuit. Thus,

$$\begin{aligned} E(x) &= \frac{1}{2}(n-1)! \left(\frac{d}{n}\right)^n \\ &\simeq \left(\frac{n}{e}\right)^n \left(\frac{d}{n}\right)^n \\ &\to \begin{cases} 0 & d < e \\ \infty & d > e \end{cases}. \end{aligned}$$

This suggests that the threshold for Hamilton circuits occurs when *d* equals Euler's constant *e.* This is not possible since the graph still has isolated vertices and is not even connected for $p = \frac{e}{n}$. Thus, the second moment argument is indeed necessary.

The actual threshold for Hamilton circuits is $\frac{1}{n}\log n$. For any *p*(*n*) asymptotically greater, *G*(*n,p*) will have a Hamilton circuit with probability one. This is the same threshold as for the disappearance of degree one vertices. Clearly a graph with a degree one vertex cannot have a Hamilton circuit. But it may seem surprising that Hamilton circuits appear as soon as degree one vertices disappear. You may ask why at the moment degree one vertices disappear there cannot be a subgraph consisting of a degree three vertex adjacent to three degree two vertices as shown in Figure 8.7. The reason is that the frequency of degree two and three vertices in the graph is very small and the probability that four such vertices would occur together in such a subgraph is too small for it to happen with nonnegligible probability.

8.3 Giant Component

Consider $G(n,p)$ for $p = \frac{1+\epsilon}{n}$ where is a constant greater than zero. We now show that with high probability, such a graph contains a *giant component*, namely a component of size Ω(*n*). Moreover, with high probability, the graph contains only one such component, and all other components are much smaller, of size only *O*(log*n*). We begin by arguing existence of a giant component.

8.3.1 Existence of a giant component

To see that with high probability the graph has a giant component, do a depth first search (dfs) on *G*(*n,p*) where $p = (1+\epsilon)/n$ with $0 < \epsilon < 1/8$. Note that it suffices to consider this range of since increasing the value of *p* only increases the probability that the graph has a giant component.

To perform the dfs, generate $\binom{n}{2}$ Bernoulli(*p*) independent random bits and answer

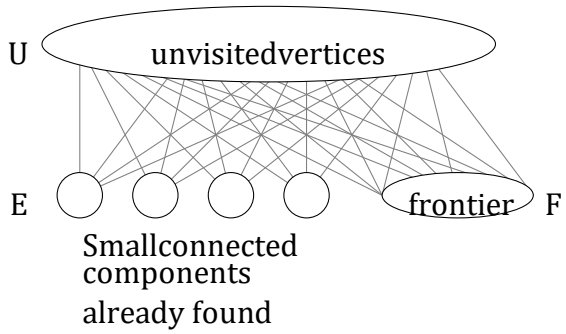


Figure 8.8: Picture after $\epsilon n^2/2$ edge queries. The potential edges from the small connected components to unvisited vertices do not exist in the graph. However, since many edges must have been found the frontier must be big and hence there is a giant component.

the t^{th} edge query according to the t^{th} bit. As the dfs proceeds, let

E = set of fully explored vertices whose exploration is complete *U* = set of unvisited vertices

F = frontier of visited and still being explored vertices .

Initially the set of fully explored vertices, *E,* and the frontier, *F* are empty and the set of unvisited vertices, *U* equals {1,2,...,*n*}. If the frontier is not empty and *u* is the active vertex of the dfs, the dfs queries each unvisited vertex in *U* until it finds a vertex *v* for which there is an edge (*u,v*) and moves *v* from *U* to the frontier and *v* becomes the active vertex. If no edge is found from *u* to an unvisited vertex in *U,* then *u* is moved from the frontier to the set of fully explored vertices *E*. If frontier is empty, the dfs moves an unvisited vertex from *U* to frontier and starts a new component. If both frontier and *U* are empty all connected components of *G* have been found. At any time all edges between the current fully explored vertices, *E,* and the current unvisited vertices, *U,* have been queried since a vertex is moved from the frontier to *E* only when there is no edge from the vertex to *U.*

Intuitively, after $\epsilon n^2/2$ edge queries a large number of edges must have been found since $p = \frac{1+\epsilon}{n}$. None of these can connect components already found with the set of unvisited vertices, and we will use this to show that with high probability the frontier must be large. Since the frontier will be in a connected component, a giant component exists with high probability. We first prove that after $\epsilon n^2/2$ edge queries the set of fully explored vertices is of size less than *n/*3*.*

Lemma 8.7 *After* $\epsilon n^2/2$ *edge queries, with high probability* |*E*| < *n/*3*.*

Proof: If not, at some $t \le \varepsilon n^2/2$, $|E| = n/3$. A vertex is added to frontier only when an edge query is answered yes. So at time t, $|F|$ is less than or equal to the sum of $\varepsilon n^2/2$ Bernoulli(p) random variables, which with high probability is at most $\varepsilon n^2 p \le n/3$. So, at t, $|U| = n - |E| - |F| \ge n/3$. Since there are no edges between fully explored vertices and unvisited vertices, $|E|$ $|U| \ge n^2/9$ edge queries must have already been answered in the negative. But $t > n^2/9$ contradicts $t \le \epsilon n^2/2 \le n^2/16$. Thus $|E| \le n/3$. ∎

The frontier vertices in the search of a connected component are all in the component being searched. Thus if at any time the frontier set has $\Omega(n)$ vertices there is a giant component.

Lemma 8.8 *After $\epsilon n^2/2$ edge queries, with high probability the set F consists of at least $\epsilon^2 n/30$ vertices.*

Proof: After $\varepsilon n^2/2$ queries, say, $|F| < \varepsilon^2 n/30$. Thus

$$|U| = n - |E| - |F| = n - \frac{n}{3} - \frac{\epsilon^2 n}{30} \ge 1$$

and so the dfs is still active. Each positive answer to an edge query so far resulted in some vertex moving from U to F, which possibly later moved to E. The expected number of yes answers so far is $p\varepsilon n^2/2 = (1 + \varepsilon)\varepsilon n/2$ and with high probability, the number of yes answers is at least $(\varepsilon n/2) + (\varepsilon^2 n/3)$. So,

$$|E| + |F| \ge \frac{\varepsilon n}{2} + \frac{\varepsilon^2 n}{3} \implies |E| \ge \frac{\varepsilon n}{2} + \frac{3\varepsilon^2 n}{10}.$$

We must have $|E|$ $|U| \le \varepsilon n^2/2$. Now, $|E||U| = |E|(n-|E|-|F|)$ increases as $|E|$ increases from $\frac{\varepsilon n}{2} + \frac{3\varepsilon^2 n}{10}$ to $n/3$, so we have

$$|E||U| \ge \left(\frac{\varepsilon n}{2} + \frac{3\varepsilon^2 n}{10}\right)\left(n - \frac{\varepsilon n}{2} - \frac{3\varepsilon^2 n}{10} - \frac{\varepsilon^2 n}{30}\right) > \frac{\varepsilon n^2}{2},$$

a contradiction. ∎

8.3.2 No other large components

We now argue that for $p = (1 + \epsilon)/n$ for constant > 0, with high probability there is only one giant component, and in fact all other components have size $O(\log n)$.

We begin with a preliminary observation. Suppose that a $G(n,p)$ graph had at least a δ probability of having two (or more) components of size $\omega(\log n)$, i.e., asymptotically greater than $\log n$. Then, there would be at least a $\delta/2$ probability of the graph having two (or more) components with $\omega(\log n)$ vertices *inside the subset* $A = \{1, 2, \ldots, \epsilon n/2\}$.
The reason is that an equivalent way to construct a graph $G(n,p)$ is to first create it in the usual way and then to randomly permute the vertices. Any component of size $\omega(\log n)$ will

with high probability after permutation have at least an /4 fraction of its vertices within the first $n/2$. Thus, it suffices to prove that with high probability at most one component has $\omega(\log n)$ vertices within the set A to conclude that with high probability the graph has only one component with $\omega(\log n)$ vertices overall.

We now prove that with high probability, a $G(n,p)$ graph for $p = (1+\epsilon)/n$ has at most one component with $\omega(\log n)$ vertices inside the set A. To do so, let B be the set of $(1-\epsilon/2)n$ vertices not in A. Now, construct the graph as follows. First, randomly flip coins of bias p to generate the edges within set A and the edges within set B. At this point, with high probability, B has at least one giant component, by the argument from Section 8.3.1, since $p = (1+\epsilon)/n \geq (1+\epsilon/4)/|B|$ for $0 < \epsilon \leq 1/2$. Let C^* be a giant component inside B. Now, flip coins of bias p to generate the edges between A and B *except* for those incident to C^*. At this point, let us name all components with $\omega(\log n)$ vertices inside A as $C_1, C_2, C_3, \ldots$. Finally, flip coins of bias p to generate the edges between A and C^*.

In the final step above, notice that with high probability, each C_i is connected to C^*. In particular, there are $\omega(n \log n)$ possible edges between any given C_i and C^*, each one of which is present with probability p. Thus the probability that this particular C_i is *not* connected to C^* is at most $(1-p)^{\omega(n \log n)} = 1/n^{\omega(1)}$. Thus, by the union bound, with high probability all such C_i are connected to C^*, and there is only one component with $\omega(\log n)$ vertices within A as desired.

8.3.3 The case of $p < 1/n$

When $p < 1/n$, then with high probability all components in $G(n,p)$ are of size $O(\log n)$. This is easiest to see by considering a variation on the above dfs that (a) begins with F containing a specific start vertex u_{start}, and then (b) when a vertex u is taken from F to explore, it pops u off of F, explores u fully by querying to find *all* edges between u and U, and then pushes the endpoints v of those edges onto F. Thus, this is like an explicit-stack version of dfs, compared to the previous recursive-call version of dfs. Let us call the exploration of such a vertex u a *step*. To make this process easier to analyze, let us say that if F ever becomes empty, we create a brand-new, fake "red vertex", connect it to each vertex in U with probability p, place the new red vertex into F, and then continue the dfs from there.

Let z_k denote the number of real (non-red) vertices discovered after k steps, not including u_{start}. For any given real vertex $u \neq u_{start}$, the probability that u is not discovered in k steps is $(1-p)^k$, and notice that these events are independent over the different vertices $u \neq u_{start}$. Therefore, the distribution of z_k is Binomial $\left(n-1, 1-(1-p)^k\right)$. Note that if $z_k < k$ then the process must have required creating a fake red vertex by step k, meaning that u_{start} is in a component of size at most k. Thus, it suffices to prove that $\text{Prob}(z_k \geq k) < 1/n^2$, for $k = c \ln n$ for a suitably large constant c, to then conclude by union bound

over choices of u_{start} that with high probability *all* vertices are in components of size at most $c\ln n$.

To prove that $\text{Prob}(z_k \geq k) < 1/n^2$ for $k = c\ln n$, we use the fact that $(1-p)^k \geq 1-pk$ so $1 - (1-p)^k \leq pk$. So, the probability that z_k is greater than or equal to k is at most the probability that a coin of bias pk flipped $n-1$ times will have at least k heads. But since $pk(n-1) \leq (1-\epsilon)k$ for some constant $\epsilon > 0$, by Chernoff bounds this probability is at most $e^{-c_0 k}$ for some constant $c_0 > 0$. When $k = c\ln n$ for a suitably large constant c, this probability is at most $1/n^2$, as desired.

8.4 Cycles and Full Connectivity

This section considers when cycles form and when the graph becomes fully connected. For both of these problems, we look at each subset of k vertices and see when they form either a cycle or when they form a connected component.

8.4.1 Emergence of Cycles

The emergence of cycles in $G(n,p)$ has a threshold when p equals to $1/n$. However, the threshold is not sharp.

Theorem 8.9 *The threshold for the existence of cycles in $G(n,p)$ is $p = 1/n$.*

Proof: Let x be the number of cycles in $G(n,p)$. To form a cycle of length k, the vertices can be selected in $\binom{n}{k}$ ways. Given the k vertices of the cycle, they can be ordered by arbitrarily selecting a first vertex, then a second vertex in one of k-1 ways, a third in one of $k-2$ ways, etc. Since a cycle and its reversal are the same cycle, divide by 2. Thus, there are $\binom{n}{k}\frac{(k-1)!}{2}$ possible cycles of length k and

$$E(x) = \sum_{k=3}^{n} \binom{n}{k} \frac{(k-1)!}{2} p^k \leq \sum_{k=3}^{n} \frac{n^k}{2k} p^k \leq \sum_{k=3}^{n} (np)^k = (np)^3 \frac{1-(np)^{n-2}}{1-np} \leq 2(np)^3,$$

provided that $np < 1/2$. When p is asymptotically less than $1/n$, then $\lim_{n\to\infty} np = 0$ and

$\lim_{n\to\infty} \sum_{k=3}^{n} (np)^k = 0$. So, as n goes to infinity, $E(x)$ goes to zero. Thus, the graph almost surely has no cycles by the first moment method. A second moment argument can be used to show that for $p = d/n$, $d > 1$, a graph will have a cycle with probability tending to one.

■

The argument above does not yield a sharp threshold since we argued that $E(x) \to 0$ only under the assumption that p is asymptotically less than $\frac{1}{n}$. A sharp threshold requires $E(x) \to 0$ for $p = d/n$, $d < 1$.

Property	Threshold
cycles $1/n$	
giant component $1/n$	
giant component $\frac{1}{2}\ln n$ + isolated vertices	$\frac{1}{2}\ln n$
connectivity, disappearance of isolated vertices	$\ln n$
diameter two	$\sqrt{\frac{2\ln n}{n}}$

Table 2: Thresholds for various properties

Consider what happens in more detail when $p = d/n$, d a constant.

$$\begin{aligned} E(x) &= \sum_{k=3}^{n} \binom{n}{k} \frac{(k-1)!}{2} p^k \\ &= \frac{1}{2} \sum_{k=3}^{n} \frac{n(n-1)\cdots(n-k+1)}{k!} (k-1)!\, p^k \\ &= \frac{1}{2} \sum_{k=3}^{n} \frac{n(n-1)\cdots(n-k+1)}{n^k} \frac{d^k}{k}. \end{aligned}$$

$E(x)$ converges if $d < 1$, and diverges if $d \geq 1$. If $d < 1$, $E(x) \leq \frac{1}{2} \sum_{k=3}^{n} \frac{d^k}{k}$ and $\lim_{n\to\infty} E(x)$ equals a constant greater than zero. If $d = 1$, $E(x) = \frac{1}{2} \sum_{k=3}^{n} \frac{n(n-1)\cdots(n-k+1)}{n^k} \frac{1}{k}$. Consider only the first log$n$ terms of the sum. Since $\frac{n}{n-i} = 1 + \frac{i}{n-i} \leq e^{i/n-i}$, it follows that $\frac{n(n-1)\cdots(n-k+1)}{n^k} \geq 1/2$. Thus,

$$E(x) \geq \frac{1}{2} \sum_{k=3}^{\log n} \frac{n(n-1)\cdots(n-k+1)}{n^k} \frac{1}{k} \geq \frac{1}{4} \sum_{k=3}^{\log n} \frac{1}{k}.$$

Then, in the limit as n goes to infinity

$$\lim_{n\to\infty} E(x) \geq \lim_{n\to\infty} \frac{1}{4} \sum_{k=3}^{\log n} \frac{1}{k} \geq \lim_{n\to\infty} (\log\log n) = \infty.$$

For $p = d/n$, $d < 1$, $E(x)$ converges to a nonzero constant. For $d > 1$, $E(x)$ converges to infinity and a second moment argument shows that graphs will have an unbounded number of cycles increasing with n.

8.4.2 Full Connectivity

As p increases from $p = 0$, small components form. At $p = 1/n$ a giant component emerges and swallows up smaller components, starting with the larger components and ending up swallowing isolated vertices forming a single connected component at $p = \frac{\ln n}{n}$, at which point the graph becomes connected. We begin our development with a technical lemma.

Lemma 8.10 *The expected number of connected components of size k in G(n,p) is at most*

$$\binom{n}{k} k^{k-2} p^{k-1} (1-p)^{kn-k^2}.$$

Proof: The probability that k vertices form a connected component consists of the product of two probabilities. The first is the probability that the k vertices are connected, and the second is the probability that there are no edges out of the component to the remainder of the graph. The first probability is at most the sum over all spanning trees of the k vertices, that the edges of the spanning tree are present. The "at most" in the lemma statement is because $G(n,p)$ may contain more than one spanning tree on these nodes and, in this case, the union bound is higher than the actual probability. There are k^{k-2} spanning trees on k nodes. See Section 12.10.5 in the appendix. The probability of all the $k - 1$ edges of one spanning tree being present is p^{k-1} and the probability that there are no edges connecting the k vertices to the remainder of the graph is $(1 - p)^{k(n-k)}$. Thus, the probability of one particular set of k vertices forming a connected component is at most $k^{k-2}p^{k-1}(1 - p)^{kn-k_2}$. Thus, the expected number of connected components of size k is at most $\binom{n}{k} k^{k-2} p^{k-1} (1-p)^{kn-k^2}$. ∎

We now prove that for $p = \frac{1}{2}\frac{\ln n}{n}$, the giant component has absorbed all small components except for isolated vertices.

Theorem 8.11 *For $p = c\frac{\ln n}{n}$ with $c > 1/2$, almost surely there are only isolated vertices and a giant component. For $c > 1$, almost surely the graph is connected.*

Proof: We prove that almost surely for $c > 1/2$, there is no connected component with k vertices for any k, $2 \le k \le n/2$. This proves the first statement of the theorem since, if there were two or more components that are not isolated vertices, both of them could not be of size greater than $n/2$. The second statement that for $c > 1$ the graph is connected then follows from Theorem 8.6 which states that isolated vertices disappear at $c = 1$.

We now show that for $p = c\frac{\ln n}{n}$, the expected number of components of size k, $2 \le k \le n/2$, is less than n^{1-2c} and thus for $c > 1/2$ there are no components, except for isolated vertices and the giant component. Let x_k be the number of connected components of size k. Substitute $p = c\frac{\ln n}{n}$ into $\binom{n}{k} k^{k-2} p^{k-1} (1-p)^{kn-k^2}$ and simplify using $\binom{n}{k} \le (en/k)^k$, $1 - p \le e^{-p}$, $k - 1 < k$, and $x = e^{\ln x}$ to get

$$E(x_k) \leq \exp\left(\ln n + k + k\ln\ln n - 2\ln k + k\ln c - ck\ln n + ck^2\frac{\ln n}{n}\right).$$

Keep in mind that the leading terms here for large k are the last two and, in fact, at $k = n$, they cancel each other so that our argument does not prove the fallacious statement for $c \geq 1$ that there is no connected component of size n, since there is. Let

$$f(k) = \ln n + k + k\ln\ln n - 2\ln k + k\ln c - ck\ln n + ck^2\frac{\ln n}{n}.$$

Differentiating with respect to k,

$$f'(k) = 1 + \ln\ln n - \frac{2}{k} + \ln c - c\ln n + \frac{2ck\ln n}{n}$$

and

$$f''(k) = \frac{2}{k^2} + \frac{2c\ln n}{n} > 0.$$

Thus, the function $f(k)$ attains its maximum over the range $[2, n/2]$ at one of the extreme points 2 or $n/2$. At $k = 2$, $f(2) \approx (1 - 2c)\ln n$ and at $k = n/2,\ f(n/2) \approx -c\frac{n}{4}\ln n$. So $f(k)$ is maximum at $k = 2$. For $k = 2$, $E(x_k) = e^{f(k)}$ is approximately $e^{(1-2c)\ln n} = n^{1-2c}$ and is geometrically falling as k increases from 2. At some point $E(x_k)$ starts to increase but never gets above $n^{-\frac{c}{4}n}$. Thus, the expected sum of the number of components of size k, for $2 \leq k \leq n/2$ is

$$E\left(\sum_{k=2}^{n/2} x_k\right) = O(n^{1-2c}).$$

This expected number goes to zero for $c > 1/2$ and the first-moment method implies that, almost surely, there are no components of size between 2 and $n/2$. This completes the proof of Theorem 8.11. ∎

8.4.3 Threshold for $O(\ln n)$ Diameter

We now show that within a constant factor of the threshold for graph connectivity, not only is the graph connected, but its diameter is $O(\ln n)$. That is, if $p > c\frac{\ln n}{n}$ for sufficiently large constant c, the diameter of $G(n,p)$ is $O(\ln n)$ with high probability.

Consider a particular vertex v. Let S_i be the set of vertices at distance i from v. We argue that as i increases, with high probability $|S_1| + |S_2| + \cdots + |S_i|$ grows by at least a factor of two, up to a size of $n/1000$. This implies that in $O(\ln n)$ steps, at least $n/1000$ vertices are connected to v. Then, there is a simple argument at the end of the proof of Theorem 8.13 that a pair of $n/1000$ sized subsets, connected to two different vertices v and w, have an edge between them with high probability.

Lemma 8.12 *Consider $G(n,p)$ for sufficiently large n with $p = c\frac{\ln n}{n}$ for any $c > 0$. Let S_i be the set of vertices at distance i from some fixed vertex v. If $|S_1|+|S_2|+\cdots+|S_i| \le n/1000$, then*

$$Prob\left(|S_{i+1}| < 2(|S_1| + |S_2| + \cdots + |S_i|)\right) \le e^{-10|S_i|}.$$

Proof: Let $|S_i| = k$. For each vertex u not in $S_1 \cup S_2 \cup \ldots \cup S_i$, the probability that u is not in S_{i+1} is $(1-p)^k$ and these events are independent. So, $|S_{i+1}|$ is the sum of $n-(|S_1|+|S_2|+\cdots+|S_i|)$ independent Bernoulli random variables, each with probability of

$$1 - (1-p)^k \ge 1 - e^{-ck\ln n/n}$$

of being one. Note that $n - (|S_1| + |S_2| + \cdots + |S_i|) \ge 999n/1000$. So,

$$E(|S_{i+1}|) \ge \frac{999n}{1000}(1 - e^{-ck\frac{\ln n}{n}}).$$

Subtracting $200k$ from each side

$$E(|S_{i+1}|) - 200k \ge \frac{n}{2}\left(1 - e^{-ck\frac{\ln n}{n}} - 400\frac{k}{n}\right).$$

Let $\alpha = \frac{k}{n}$ and $f(\alpha) = 1 - e^{-c\alpha\ln n} - 400\alpha$. By differentiation $f^{00}(\alpha) \le 0$, so f is concave and the minimum value of f over the interval $[0,1/1000]$ is attained at one of the end points. It is easy to check that both $f(0)$ and $f(1/1000)$ are greater than or equal to zero for sufficiently large n. Thus, f is nonnegative throughout the interval proving that $E(|S_{i+1}|) \ge 200|S_i|$. The lemma follows from Chernoff bounds. ■

Theorem 8.13 *For $p \ge c\ln n/n$, where c is a sufficiently large constant, almost surely, $G(n,p)$ has diameter $O(\ln n)$.*

Proof: By Corollary 8.2, almost surely, the degree of every vertex is $\Omega(np) = \Omega(\ln n)$, which is at least $20\ln n$ for c sufficiently large. Assume that this holds. So, for a fixed vertex v, S_1 as defined in Lemma 8.12 satisfies $|S_1| \ge 20\ln n$.

Let i_0 be the least i such that $|S_1|+|S_2|+\cdots+|S_i| > n/1000$. From Lemma 8.12 and the union bound, the probability that for some $i, 1 \le i \le i_0-1$, $|S_{i+1}| < 2(|S_1|+|S_2|+\cdots+|S_i|)$ is at most $\sum_{k=20\ln n}^{n/1000} e^{-10k} \le 1/n^4$. So, with probability at least $1 - (1/n^4)$, each S_{i+1} is at least double the sum of the previous S_j's, which implies that in $O(\ln n)$ steps, $i_0 + 1$ is reached.

Consider any other vertex w. We wish to find a short $O(\ln n)$ length path between v and w. By the same argument as above, the number of vertices at distance $O(\ln n)$ from w is at least $n/1000$. To complete the argument, either these two sets intersect in which case we have found a path from v to w of length $O(\ln n)$ or they do not intersect. In the latter case, with high probability there is some edge between them. For a pair of disjoint sets of size at least $n/1000$, the probability that none of the possible $n^2/10^6$ or more edges between them is present is at most $(1-p)^{n^2/10^6} = e^{-\Omega(n\ln n)}$. There are at most 2^{2n} pairs of such sets and so the probability that there is some such pair with no edges is $e^{-\Omega(n\ln n)+O(n)} \to 0$. Note that there is no conditioning problem since we are arguing this for every pair of such sets.

Think of whether such an argument made for just the n subsets of vertices, which are vertices at distance at most $O(\ln n)$ from a specific vertex, would work. ■

8.5 Phase Transitions for Increasing Properties

For many graph properties such as connectivity, having no isolated vertices, having a cycle, etc., the probability of a graph having the property increases as edges are added to the graph. Such a property is called an increasing property. Q is an *increasing property* of graphs if when a graph G has the property, any graph obtained by adding edges to G must also have the property. In this section we show that any increasing property has a threshold, although not necessarily a sharp one.

The notion of increasing property is defined in terms of adding edges. The following intuitive lemma proves that if Q is an increasing property, then increasing p in $G(n,p)$ increases the probability of the property Q.

Lemma 8.14 *If Q is an increasing property of graphs and $0 \le p \le q \le 1$, then the probability that $G(n,q)$ has property Q is greater than or equal to the probability that $G(n,p)$ has property Q.*

Proof: This proof uses an interesting relationship between $G(n,p)$ and $G(n,q)$. Generate $G(n,q)$ as follows. First generate $G(n,p)$. This means generating a graph on n vertices with edge probabilities p. Then, independently generate another graph $G\left(n, \frac{q-p}{1-p}\right)$ and take the union by including an edge if either of the two graphs has the edge. Call the resulting graph H. The graph H has the same distribution as $G(n,q)$. This follows since the probability that an edge is in H is $p + (1-p)\frac{q-p}{1-p} = q,$ and, clearly, the edges of H are independent. The lemma follows since whenever $G(n,p)$ has the property Q, H also has the property Q. ■

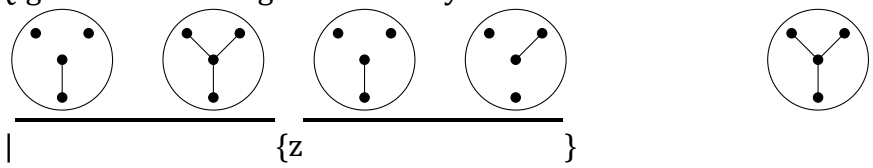
We now introduce a notion called *replication*. An m-fold replication of $G(n,p)$ is a random graph obtained as follows. Generate m independent copies of $G(n,p)$ on the same set of vertices. Include an edge in the m-fold replication if the edge is in any one of the m copies of $G(n,p)$. The resulting random graph has the same distribution as $G(n,q)$ where $q = 1 - (1-p)^m$ since the probability that a particular edge is not in the m-fold replication is the product of probabilities that it is not in any of the m copies of $G(n,p)$. If the m-fold replication of $G(n,p)$ does not have an increasing property Q, then none of the m copies of $G(n,p)$ has the property. The converse is not true. If no copy has the property, their union may have it. Since Q is an increasing property and

$q = 1 - (1-p)^m \le 1 - (1 - mp) = mp$

$$\text{Prob } G(n,mp) \text{ has } Q) \ge \text{Prob } G(n,q) \text{ has } Q \tag{8.3}$$

We now show that every increasing property Q has a phase transition. The transition occurs at the point $p(n)$ at which the probability that $G(n,p(n))$ has property Q is $\frac{1}{2}$. We

will prove that for any function asymptotically less then $p(n)$ that the probability of having property Q goes to zero as n goes to infinity.



If any graph has three or more edges, then the m-fold replication has three or more edges.
Even if no graph has three or more edges, the m-fold replication might have three or more edges.

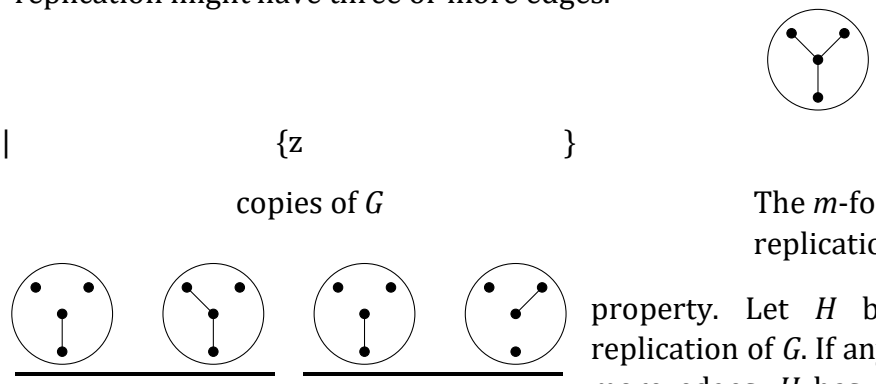


Figure 8.9: The property that G has three or more edges is an increasing property. Let H be the m-fold replication of G. If any copy of G has three or more edges, H has three or more edges. However, H can have three or more edges even if no copy of G has three or more edges.

Theorem 8.15 *Each increasing property Q of $G(n,p)$ has a phase transition at $p(n)$, where for each n, $p(n)$ is the minimum real number a_n for which the probability that $G(n,a_n)$ has property Q is 1/2.*

Proof: Let $p_0(n)$ be any function such that

$$\lim_{n\to\infty} \frac{p_0(n)}{p(n)} = 0$$

.

We assert that almost surely $G(n,p_0)$ does not have the property Q. Suppose for contradiction, that this is not true. That is, the probability that $G(n,p_0)$ has the property Q does not converge to zero. By the definition of a limit, there exists $\varepsilon > 0$ for which the probability that $G(n,p_0)$ has property Q is at least ε on an infinite set I of n. Let $m = \mathrm{d}(1/\varepsilon)\mathrm{e}$. Let $G(n,q)$ be the m-fold replication of $G(n,p_0)$. The probability that $G(n,q)$ does not have Q is at most $(1-\varepsilon)^m \le e^{-1} \le 1/2$ for all $n \in I$. For these n, by

(11.4)

$$\text{Prob}(G(n,mp_0) \text{ has } Q) \ge \text{Prob}(G(n,q) \text{ has } Q) \ge 1/2.$$

Since $p(n)$ is the minimum real number a_n for which the probability that $G(n,a_n)$ has property Q is 1/2, it must be that $mp_0(n) \geq p(n)$. This implies that $\frac{p_0(n)}{p(n)}$ is at least $1/m$ infinitely often, contradicting the hypothesis that $\lim_{n\to\infty} \frac{p_0(n)}{p(n)} = 0$.

A symmetric argument shows that for any $p_1(n)$ such that $\lim_{n\to\infty} \frac{p(n)}{p_1(n)} = 0$, $G(n, p_1)$ almost surely has property Q. ■

8.6 Branching Processes

A *branching process* is a method for creating a random tree. Starting with the root node, each node has a probability distribution for the number of its children. The root of the tree is a parent and its descendants are the children with their descendants being the grandchildren. The children of the root are the first generation, their children the second generation, and so on. Branching processes have obvious applications in population studies.

We analyze a simple case of a branching process where the distribution of the number of children at each node in the tree is the same. The basic question asked is what is the probability that the tree is finite, i.e., the probability that the branching process dies out? This is called the *extinction probability*.

Our analysis of the branching process will give the probability of extinction, as well as the expected size of the components conditioned on extinction.

An important tool in our analysis of branching processes is the generating function. The generating function for a nonnegative integer valued random variable y is

$f(x) = \sum_{i=0}^{\infty} p_i x^i$ where p_i is the probability that y equals i. The reader not familiar with generating functions should consult Section 12.9 of the appendix.

Let the random variable z_j be the number of children in the j^{th} generation and let $f_j(x)$ be the generating function for z_j. Then $f_1(x) = f(x)$ is the generating function for the first generation where $f(x)$ is the generating function for the number of children at a node in the tree. The generating function for the 2^{nd} generation is $f_2(x) = f(f(x))$. In general, the generating function for the j+1^{st} generation is given by $f_{j+1}(x) = f_j(f(x))$. To see this, observe two things.

First, the generating function for the sum of two identically distributed integer valued random variables x_1 and x_2 is the square of their generating function

$$f^2(x) = p_0^2 + (p_0p_1 + p_1p_0)\,x + (p_0p_2 + p_1p_1 + p_2p_0)\,x^2 + \cdots.$$

For $x_1 + x_2$ to have value zero, both x_1 and x_2 must have value zero, for $x_1 + x_2$ to have value one, exactly one of x_1 or x_2 must have value zero and the other have value one, and so on. In general, the generating function for the sum of i independent random variables, each with generating function $f(x)$, is $f^i(x)$.

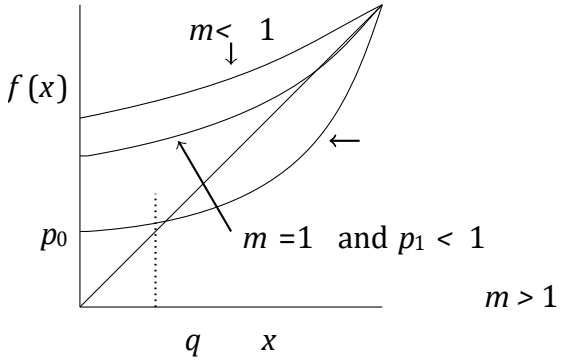


Figure 8.10: Illustration of the root of equation $f(x) = x$ in the interval [0,1).

The second observation is that the coefficient of x^i in $f_j(x)$ is the probability of there being i children in the j^{th} generation. If there are i children in the j^{th} generation, the number of children in the $j + 1^{st}$ generation is the sum of i independent random variables each with generating function $f(x)$. Thus, the generating function for the $j+1^{st}$ generation, given i children in the j^{th} generation, is $f^i(x)$. The generating function for the $j + 1^{st}$ generation is given by

$$f_{j+1}(x) = \sum_{i=0}^{\infty} \text{Prob}(z_j = i) f^i(x).$$

If $f_j(x) = \sum_{i=0}^{\infty} a_i x^i$, then f_{j+1} is obtained by substituting $f(x)$ for x in $f_j(x)$.

Since $f(x)$ and its iterates, $f_2, f_3, \ldots$, are all polynomials in x with nonnegative coefficients, $f(x)$ and its iterates are all monotonically increasing and convex on the unit interval. Since the probabilities of the number of children of a node sum to one, if $p_0 < 1$, some coefficient of x to a power other than zero in $f(x)$ is nonzero and $f(x)$ is strictly increasing.

Let q be the probability that the branching process dies out. If there are i children in the first generation, then each of the i subtrees must die out and this occurs with probability q^i. Thus, q equals the summation over all values of i of the product of the probability of i children times the probability that i subtrees will die out. This gives

$q = \sum_{i=0}^{\infty} p_i q^i$. Thus, q is the root of $x = \sum_{i=0}^{\infty} p_i x^i$, that is $x = f(x)$.

This suggests focusing on roots of the equation $f(x) = x$ in the interval [0,1]. The value $x = 1$ is always a root of the equation $f(x) = x$ since $f(1) = \sum_{i=0}^{\infty} p_i = 1$. When is there a smaller nonnegative root? The derivative of $f(x)$ at $x = 1$ is $f'(1) = p_1 + 2p_2 + 3p_3 + \cdots$. Let $m = f'(1)$. Thus, m is the expected number of children of a node. If $m > 1$, one might expect the tree to grow forever, since each node at time j is expected to have more

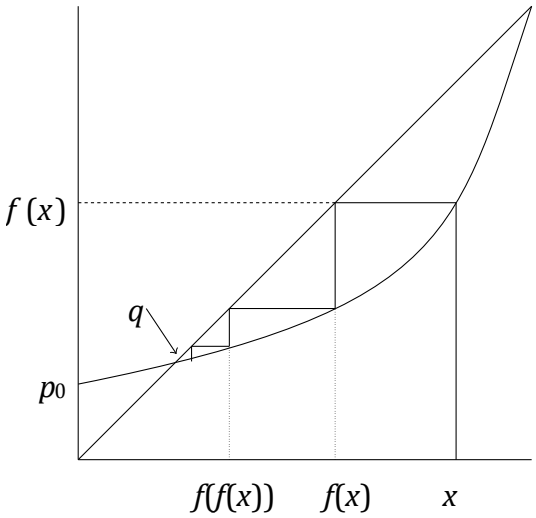


Figure 8.11: Illustration of convergence of the sequence of iterations $f_1(x), f_2(x), \ldots$ to q.

than one child. But this does not imply that the probability of extinction is zero. In fact, if $p_0 > 0$, then with positive probability, the root will have no children and the process will become extinct right away. Recall that for $G(n, \frac{d}{n})$, the expected number of children is d, so the parameter m plays the role of d.

If $m < 1$, then the slope of $f(x)$ at $x = 1$ is less than one. This fact along with convexity of $f(x)$ implies that $f(x) > x$ for x in [0,1) and there is no root of $f(x) = x$ in the interval [0,1).

If $m = 1$ and $p_1 < 1$, then once again convexity implies that $f(x) > x$ for $x \in [0,1)$ and there is no root of $f(x) = x$ in the interval [0,1). If $m = 1$ and $p_1 = 1$, then $f(x)$ is the straight line $f(x) = x$.

If $m > 1$, then the slope of $f(x)$ is greater than the slope of x at $x = 1$. This fact, along with convexity of $f(x)$, implies $f(x) = x$ has a unique root in [0,1). When $p_0 = 0$, the root is at $x = 0$.

Let q be the smallest nonnegative root of the equation $f(x) = x$. For $m < 1$ and for $m=1$ and $p_0 < 1$, q equals one and for $m > 1$, q is strictly less than one. We shall see that the value of q is the *extinction probability* of the branching process and that $1 - q$ is the *immortality probability*. That is, q is the probability that for some j, the number of children in the j^{th} generation is zero. To see this, note that for $m > 1$, $\lim_{j\to\infty} f_j(x) = q$ for $0 \le x < 1$. Figure 8.11 illustrates the proof which is given in Lemma 8.16. Similarly note that when $m < 1$ or $m = 1$ with $p_0 < 1$, $f_j(x)$ approaches one as j approaches infinity.

Lemma 8.16 *Assume $m > 1$. Let q be the unique root of $f(x)=x$ in $[0,1)$. In the limit as j goes to infinity, $f_j(x) = q$ for x in $[0,1)$.*

Proof: If $0 \le x \le q$, then $x < f(x) \le f(q)$ and iterating this inequality

$$x < f_1(x) < f_2(x) < \cdots < f_j(x) < f(q) = q.$$

Clearly, the sequence converges and it must converge to a fixed point where $f(x) = x$. Similarly, if $q \le x < 1$, then $f(q) \le f(x) < x$ and iterating this inequality

$$x > f_1(x) > f_2(x) > \cdots > f_j(x) > f(q) = q.$$

In the limit as j goes to infinity $f_j(x) = q$ for all x, $0 \le x < 1$. That is

$$\lim_{j\to\infty} f_j(x) = q + 0x + 0x^2 + \cdots$$

and there are no children with probability q and no finite number of children with probability zero. ■

Recall that $f_j(x)$ is the generating function $\sum_{i=0}^{\infty} \text{Prob}(z_j = i)x^i$. The fact that in the limit the generating function equals the constant q, and is not a function of x, says that $\text{Prob}(z_j = 0) = q$ and $\text{Prob}(z_j = i) = 0$ for all finite nonzero values of i. The remaining probability is the probability of a nonfinite component. Thus, when $m > 1$, q is the extinction probability and $1-q$ is the probability that z_j grows without bound.

Theorem 8.17 *Consider a tree generated by a branching process. Let $f(x)$ be the generating function for the number of children at each node.*

1. *If the expected number of children at each node is less than or equal to one, then the probability of extinction is one unless the probability of exactly one child is one.*
2. *If the expected number of children of each node is greater than one, then the probability of extinction is the unique solution to $f(x) = x$ in $[0,1)$.*

Proof: Let p_i be the probability of i children at each node. Then $f(x) = p_0 + p_1x + p_2x^2 + \cdots$ is the generating function for the number of children at each node and $f'(1) = p_1 + 2p_2 + 3p_3$

+ ⋯ is the slope of $f(x)$ at $x = 1$. Observe that $f^0(1)$ is the expected number of children at each node.

Since the expected number of children at each node is the slope of $f(x)$ at $x = 1$, if the expected number of children is less than or equal to one, the slope of $f(x)$ at $x = 1$ is less than or equal to one and the unique root of $f(x) = x$ in (0,1] is at $x = 1$ and the probability of extinction is one unless $f^0(1) = 1$ and $p_1 = 1$. If $f^0(1) = 1$ and $p_1 = 1$, $f(x) = x$ and the tree is an infinite degree one chain. If the slope of $f(x)$ at $x = 1$ is greater than one, then the probability of extinction is the unique solution to $f(x) = x$ in
[0,1). ∎

A branching process can be viewed as the process of creating a component in an infinite graph. In a finite graph, the probability distribution of descendants is not a constant as more and more vertices of the graph get discovered.

The simple branching process defined here either dies out or goes to infinity. In biological systems there are other factors, since processes often go to stable populations. One possibility is that the probability distribution for the number of descendants of a child depends on the total population of the current generation.

Expected size of extinct families

We now show that the expected size of an extinct family is finite, provided that m 6= 1. Note that at extinction, the size must be finite. However, the expected size at extinction could conceivably be infinite, if the probability of dying out did not decay fast enough. For example, suppose that with probability $\frac{1}{2}$ it became extinct with size 3, with probability $\frac{1}{4}$ it became extinct with size 9, with probability $\frac{1}{8}$ it became extinct with size 27, etc. In such a case the expected size at extinction would be infinite even though the process dies out with probability one. We now show this does not happen.

Lemma 8.18 *If the slope $m = f^0(1)$ does not equal one, then the expected size of an extinct family is finite. If the slope m equals one and $p_1 = 1$, then the tree is an infinite degree one chain and there are no extinct families. If m=1 and $p_1 < 1$, then the expected size of the extinct family is infinite.*

Proof: Let z_i be the random variable denoting the size of the i^{th} generation and let q be the probability of extinction. The probability of extinction for a tree with k children in the first generation is q^k since each of the k children has an extinction probability of q. Note that the expected size of z_1, the first generation, over extinct trees will be smaller than the expected size of z_1 over all trees since when the root node has a larger number of children than average, the tree is more likely to be infinite.

By Bayes rule

$$\text{Prob}(z_1 = k|\text{extinction}) = \text{Prob}(z_1 = k)\ \frac{\text{Prob}(\text{extinction}|z_1 = k)}{\text{Prob}(\text{extinction})} = p_k \frac{q^k}{q} = p_k q^{k-1}.$$

Knowing the probability distribution of z_1 given extinction, allows us to calculate the expected size of z_1 given extinction.

$$E(z_1|\text{extinction}) = \sum_{k=0}^{\infty} k p_k q^{k-1} = f'(q).$$

We now prove, using independence, that the expected size of the i^{th} generation given extinction is

$$E(z_i|\text{extinction}) = \left(f'(q)\right)^i.$$

For $i = 2$, z_2 is the sum of z_1 independent random variables, each independent of the random variable z_1. So, $E(z_2|z_1 = j$ and extinction) = $E($ sum of j copies of $z_1|$extinction) = $jE(z_1|$extinction). Summing over all values of j

$$E(z_2|\text{extinction}) = \sum_{j=1}^{\infty} E(z_2|z_1 = j \text{ and extinction})\text{Prob}(z_1 = j|\text{extinction})$$

$$= \sum_{j=1}^{\infty} jE(z_1|\text{extinction})\text{Prob}(z_1 = j|\text{extinction})$$

$$= E(z_1|\text{extinction})\sum_{j=1}^{\infty} j\text{Prob}(z_1 = j|\text{extinction}) = E^2(z_1|\text{extinction}).$$

Since $E(z_1|\text{extinction}) = f^0(q)$, $E(z_2|\text{extinction}) = (f^0(q))^2$. Similarly, $E(z_i|\text{extinction}) = (f^0(q))^i$. The expected size of the tree is the sum of the expected sizes of each generation. That is,

$$\text{Expected size of tree given extinction} = \sum_{i=0}^{\infty} E(z_i|\text{extinction}) = \sum_{i=0}^{\infty} (f'(q))^i = \frac{1}{1 - f_0(q)}.$$

Thus, the expected size of an extinct family is finite since $f^0(q) < 1$ provided $m \neq 1$.

The fact that $f^0(q) < 1$ is illustrated in Figure 8.10. If $m < 1$, then $q = 1$ and $f^0(q) = m$ is less than one. If $m > 1$, then $q \in [0,1)$ and again $f^0(q) < 1$ since q is the solution to $f(x) = x$ and $f^0(q)$ must be less than one for the curve $f(x)$ to cross the line x. Thus, for $m < 1$ or $m > 1$,

$f^0(q) < 1$ and the expected tree size of $\frac{1}{1-f'(q)}$ is finite. For $m=1$ and $p_1 < 1$, one has $q=1$ and thus $f^0(q) = 1$ and the formula for the expected size of the tree diverges. ∎

8.7 CNF-SAT

Phase transitions occur not only in random graphs, but in other random structures as well. An important example is that of satisfiability of Boolean formulas in conjunctive normal form. A conjunctive normal form (CNF) formula over n variables $x_1,\ldots,x_n$ is an AND of ORs of *literals*, where a literal is a variable or its negation. For example, the following is a CNF formula over the variables $\{x_1,x_2,x_3,x_4\}$:

$$(x_1 \vee \bar{x}_2 \vee x_3)(x_2 \vee \bar{x}_4)(x_1 \vee x_4)(x_3 \vee x_4)(x_2 \vee \bar{x}_3 \vee x_4).$$

Each OR of literals is called a *clause*; for example, the above formula has five clauses. A *k-CNF* formula is a CNF formula in which each clause has size at most k, so the above formula is a 3-CNF formula. An assignment of true/false values to variables is said to *satisfy* a CNF formula if it satisfies every clause in it. Setting all variables to true satisfies the above CNF formula, and in fact this formula has multiple satisfying assignments. A formula is said to be *satisfiable* it there exists at least one assignment of truth values to variables that satisfies it.

Many important problems can be converted into questions of finding satisfying assignments of CNF formulas. Indeed, the CNF-SAT problem of whether a given CNF formula is satisfiable is *NP-Complete*, meaning that any problem in the class NP can be converted into it. As a result, it is believed to be highly unlikely that there will ever exist an efficient algorithm for worst-case instances. However, there are solvers that turn out to work very well in practice on instances arising from a wide range of applications. There is also substantial structure and understanding of the satisfiability of *random* CNF formulas. The next two sections discuss each in turn.

8.7.1 SAT-solvers in practice

While the SAT problem is NP-complete, a number of algorithms have been developed that perform extremely well in practice on SAT formulas arising in a range of applications. Such applications include hardware and software verification, creating action plans for robots and robot teams, solving combinatorial puzzles, and even proving mathematical theorems.

Broadly, there are two classes of solvers: *complete* solvers and *incomplete* solvers. Complete solvers are guaranteed to find a satisfying assignment whenever one exists; if they do not return a solution, then you know the formula is not satisfiable. Complete solvers are often based on some form of recursive tree search. Incomplete solvers instead make a "best effort"; they are typically based on some local-search heuristic, and they may

fail to output a solution even when a formula is satisfiable. However, they are typically much faster than complete solvers.

An example of a complete solver is the following DPLL (Davis-Putnam-LogemannLoveland) style procedure. First, if there are any variables x_i that never appear in negated form in any clause, then set those variables to true and delete clauses where the literal x_i appears. Similarly, if there are any x_i that *only* appear in negated form, then set those variables to false and delete clauses where the literal $\neg x_i$ appears. Second, if there are any clauses that have only one literal in them (such clauses are called unit clauses), then set that literal as needed to satisfy the clause. E.g., if the clause was "($\neg x_3$)" then one would set x_3 to false. Then remove that clause along with any other clause containing that literal, and shrink any clause containing the negation of that literal (e.g., a clause such as ($x_3 \vee x_4$) would now become just (x_4), and one would then run this rule again on this clause). Finally, if neither of the above two cases applies, then one chooses some literal and recursively tries both settings for it. Specifically, choose some literal ` and recursively check if the formula is satisfiable conditioned on setting ` to true; if the answer is "yes" then we are done, but if the answer is "no" then recursively check if the formula is satisfiable conditioned on setting ` to false. Notice that this procedure is guaranteed to find a satisfying assignment whenever one exists.

An example of an incomplete solver is the following local-search procedure called *Walksat*. Walksat begins with a random assignment of truth-values to variables. If this happens to satisfy the formula, then it outputs success. If not, then it chooses some unsatisfied clause C at random. If C contains some variable x_i whose truth-value can be flipped (causing C to be satisfied) without causing any *other* clause to be unsatisfied, then x_i's truth-value is flipped. Otherwise, Walksat either (a) flips the truth-value of the variable in C that causes the *fewest* other clauses to become unsatisfied, or else (b) flips the truth-value of a *random* x_i in C; the choice of whether to perform (a) or (b) is determined by flipping a coin of bias p. Thus, Walksat is performing a kind of random walk in the space of truth-assignments, hence the name. Walksat also has two time-thresholds T_{flips} and $T_{restarts}$. If the above procedure has not found a satisfying assignment after T_{flips} flips, it then restarts with a fresh initial random assignment and tries again; if that entire process has not found a satisfying assignment after $T_{restarts}$ restarts, then it outputs "no assignment found".

The above solvers are just two simple examples. Due to the importance of the CNFSAT problem, development of faster SAT-solvers is an active area of computer science research. SAT-solving competitions are held each year, and solvers are routinely being used to solve challenging verification, planning, and scheduling problems.

8.7.2 Phase Transitions for CNF-SAT

We now consider the question of phase transitions in the satisfiability of *random k*CNF formulas.

Generate a random CNF formula *f* with *n* variables, *m* clauses, and *k* literals per clause, where recall that a literal is a variable or its negation. Specifically, each clause in *f* is selected independently at random from the set of all $\binom{n}{k}2^k$ possible clauses of size *k*. Equivalently, to generate a clause, choose a random set of *k* distinct variables, and then for each of those variables choose to either negate it or not with equal probability. Here, the number of variables *n* is going to infinity, *m* is a function of *n,* and *k* is a fixed constant. A reasonable value to think of for *k* is *k* = 3. Unsatisfiability is an increasing property since adding more clauses preserves unsatisfiability. By arguments similar to Section 8.5, there is a phase transition, i.e., a function $m(n)$ such that if $m_1(n)$ is $o(m(n))$, a random formula with $m_1(n)$ clauses is, almost surely, satisfiable and for $m_2(n)$ with $m_2(n)/m(n) \to \infty$, a random formula with $m_2(n)$ clauses is, almost surely, unsatisfiable. It has been conjectured that there is a constant r_k independent of *n* such that $r_k n$ is a sharp threshold.

Here we derive upper and lower bounds on r_k. It is relatively easy to get an upper bound on r_k. A fixed truth assignment satisfies a random *k* clause with probability $1 - \frac{1}{2^k}$ because of the 2^k truth assignments to the *k* variables in the clause, only one fails to satisfy the clause. Thus, with probability $\frac{1}{2^k}$, the clause is not satisfied, and with probability $1 - \frac{1}{2^k}$, the clause is satisfied. Let $m = cn$. Now, cn independent clauses are all satisfied by the fixed assignment with probability $\left(1 - \frac{1}{2^k}\right)^{cn}$. Since there are 2^n truth assignments, the expected number of satisfying assignments for a formula with cn clauses is $2^n \left(1 - \frac{1}{2^k}\right)^{cn}$. If $c = 2^k \ln 2$, the expected number of satisfying assignments is

$$2^n \left(1 - \tfrac{1}{2^k}\right)^{n 2^k \ln 2}.$$

$\left(1 - \frac{1}{2^k}\right)^{2^k}$ is at most $1/e$ and approaches $1/e$ in the limit. Thus,

$$2^n \left(1 - \tfrac{1}{2^k}\right)^{n 2^k \ln 2} \leq 2^n e^{-n \ln 2} = 2^n 2^{-n} = 1.$$

For $c > 2^k \ln 2$, the expected number of satisfying assignments goes to zero as $n \to \infty$. Here the expectation is over the choice of clauses which is random, not the choice of a truth assignment. From the first moment method, it follows that a random formula with cn clauses is almost surely not satisfiable. Thus, $r_k \leq 2^k \ln 2$.

The other direction, showing a lower bound for r_k, is not that easy. From now on, we focus only on the case *k* = 3. The statements and algorithms given here can be extended to $k \geq 4$, but with different constants. It turns out that the second moment method cannot be directly applied to get a lower bound on r_3 because the variance is too high. A simple algorithm, called the Smallest Clause Heuristic (abbreviated SC), yields a satisfying

assignment with probability tending to one if $c < \frac{2}{3}$, proving that $r_3 \geq \frac{2}{3}$. Other more difficult to analyze algorithms, push the lower bound on r_3 higher.

The Smallest Clause Heuristic repeatedly executes the following. Assign true to a random literal in a random shortest clause and delete the clause since it is now satisfied. In more detail, pick at random a 1-literal clause, if one exists, and set that literal to true. If there is no 1-literal clause, pick a 2-literal clause, select one of its two literals and set the literal to true. Otherwise, pick a 3-literal clause and a literal in it and set the literal to true. If we encounter a 0-length clause, then we have failed to find a satisfying assignment; otherwise, we have found one.

A related heuristic, called the Unit Clause Heuristic, selects a random clause with one literal, if there is one, and sets the literal in it to true. Otherwise, it picks a random as yet unset literal and sets it to true. Another variation is the "pure literal" heuristic. It sets a random "pure literal", a literal whose negation does not occur in any clause, to true, if there are any pure literals; otherwise, it sets a random literal to true.

When a literal *w* is set to true, all clauses containing *w* are deleted, since they are satisfied, and $\bar{w}$ is deleted from any clause containing $\bar{w}$. If a clause is reduced to length zero (no literals), then the algorithm has failed to find a satisfying assignment to the formula. The formula may, in fact, be satisfiable, but the algorithm has failed.

Example: Consider a 3-CNF formula with *n* variables and *cn* clauses. With *n* variables there are 2*n* literals, since a variable and its complement are distinct literals. The expected number of times a literal occurs is calculated as follows. Each clause has three literals. Thus, each of the 2*n* different literals occurs $\frac{(3cn)}{2n} = \frac{3}{2}c$ times on average. Suppose $c = 5$. Then each literal appears 7.5 times on average. If one sets a literal to true, one would expect to satisfy 7.5 clauses. However, this process is not repeatable since after setting a literal to true there is conditioning so that the formula is no longer random. ■

Theorem 8.19 *If the number of clauses in a random 3-CNF formula grows as cn where c is a constant less than 2/3, then with probability* $1 - o(1)$*, the Shortest Clause (SC) Heuristic finds a satisfying assignment.*

The proof of this theorem will take the rest of the section. A general impediment to proving that simple algorithms work for random instances of many problems is conditioning. At the start, the input is random and has properties enjoyed by random instances. But, as the algorithm is executed, the data is no longer random; it is conditioned on the steps of the algorithm so far. In the case of SC and other heuristics for finding a satisfying assignment for a Boolean formula, the argument to deal with conditioning is relatively simple.

We supply some intuition before giving the proof. Imagine maintaining a queue of 1 and 2-clauses. A 3-clause enters the queue when one of its literals is set to false and it becomes a 2-clause. SC always picks a 1 or 2-clause if there is one and sets one of its literals to true. At any step when the total number of 1 and 2-clauses is positive, one of the clauses is removed from the queue. Consider the arrival rate, that is, the expected number of arrivals into the queue at a given time *t*. For a particular clause to arrive into the queue at time *t* to become a 2-clause, it must contain the negation of the literal being set to true at time *t*. It can contain any two other literals not yet set. The number of such clauses is $\binom{n-t}{2}2^2$. So, the probability that a particular clause arrives in the queue at time *t* is at most

$$\frac{\binom{n-t}{2}2^2}{\binom{n}{3}2^3} \leq \frac{3}{2(n-2)}.$$

Since there are *cn* clauses in total, the arrival rate is $\frac{3c}{2}$, which for $c < 2/3$ is a constant strictly less than one. The arrivals into the queue of different clauses occur independently (Lemma 8.20), the queue has arrival rate strictly less than one, and the queue loses one or more clauses whenever it is nonempty. This implies that the queue never has too many clauses in it. A slightly more complicated argument will show that no clause remains as a 1 or 2-clause for $\omega(\ln n)$ steps (Lemma 8.21). This implies that the probability of two contradictory 1-length clauses, which is a precursor to a 0-length clause, is very small.

Lemma 8.20 *Let T_i be the first time that clause i turns into a 2-clause. T_i is ∞ if clause i gets satisfied before turning into a 2-clause. The T_i are mutually independent over the randomness in constructing the formula and the randomness in SC, and for any t,*

$$\underset{Prob}{(T_i = t)} \leq \frac{3}{2(n-2)}.$$

Proof: For the proof, generate the clauses in a different way. The important thing is that the new method of generation, called the method of "deferred decisions", results in the same distribution of input formulae as the original. The method of deferred decisions is tied in with the SC algorithm and works as follows. At any time, the length of each clause (number of literals) is all that we know; we have not yet picked which literals are in each clause. At the start, every clause has length three and SC picks one of the clauses uniformly at random. Now, SC wants to pick one of the three literals in that clause to set to true, but we do not know which literals are in the clause. At this point, we pick uniformly at random one of the $2n$ possible literals. Say for illustration, we picked $\bar{x}_{102}$. The literal $\bar{x}_{102}$ is placed in the clause and set to true. The literal x_{102} is set to false. We must also deal with occurrences of the literal or its negation in all other clauses, but again, we do not know which clauses have such an occurrence. We decide that now. For each clause, independently, with probability $3/n$ include either the literal $\bar{x}_{102}$ or its negation x_{102}, each with probability 1/2. In the case that we included $\bar{x}_{102}$ (the literal we had set to true), the clause is now deleted, and if we included x_{102} (the literal we had set to false), we decrease the residual length of the clause by one.

At a general stage, suppose the fates of i variables have already been decided and $n - i$ remain. The residual length of each clause is known. Among the clauses that are not yet satisfied, choose a random shortest length clause. Among the $n - i$ variables remaining, pick one uniformly at random, then pick it or its negation as the new literal. Include this literal in the clause thereby satisfying it. Since the clause is satisfied, the algorithm deletes it. For each other clause, do the following. If its residual length is l, decide with probability $l/(n - i)$ to include the new variable in the clause and if so with probability $1/2$ each, include it or its negation. If the literal that was set to true is included in a clause, delete the clause as it is now satisfied. If its negation is included in a clause, then just delete the literal and decrease the residual length of the clause by one.

Why does this yield the same distribution as the original one? First, observe that the order in which the variables are picked by the method of deferred decisions is independent of the clauses; it is just a random permutation of the n variables. Look at any one clause. For a clause, we decide in order whether each variable or its negation is in the clause. So for a particular clause and a particular triple $i,j,$ and k with $i < j < k$, the probability that the clause contains the i^{th}, the j^{th}, and k^{th} literal (or their negations) in the order determined by deferred decisions is:

$$\left(1-\tfrac{3}{n}\right)\left(1-\tfrac{3}{n-1}\right)\cdots\left(1-\tfrac{3}{n-i+2}\right)\tfrac{3}{n-i+1}$$
$$\left(1-\tfrac{2}{n-i}\right)\left(1-\tfrac{2}{n-i-1}\right)\cdots\left(1-\tfrac{2}{n-j+2}\right)\tfrac{2}{n-j+1}$$
$$\left(1-\tfrac{1}{n-j}\right)\left(1-\tfrac{1}{n-j-1}\right)\cdots\left(1-\tfrac{1}{n-k+2}\right)\tfrac{1}{n-k+1}=\tfrac{3}{n(n-1)(n-2)},$$

where the $(1 - \cdots)$ factors are for not picking the current variable or negation to be included and the others are for including the current variable or its negation. Independence among clauses follows from the fact that we have never let the occurrence or nonoccurrence of any variable in any clause influence our decisions on other clauses.

Now, we prove the lemma by appealing to the method of deferred decisions to generate the formula. $T_i = t$ if and only if the method of deferred decisions does not put the current literal at steps $1,2,\ldots,t-1$ into the i^{th} clause, but puts the negation of the literal at step t into it. Thus, the probability is precisely

$$\tfrac{1}{2}\left(1-\tfrac{3}{n}\right)\left(1-\tfrac{3}{n-1}\right)\cdots\left(1-\tfrac{3}{n-t+2}\right)\tfrac{3}{n-t+1}\le\tfrac{3}{2(n-2)},$$

as claimed. Clearly the T_i are independent since again deferred decisions deal with different clauses independently. ■

Lemma 8.21 *There exists a constant c_2 such that with probability $1 - o(1)$, no clause remains a 2 or 1-clause for more than $c_2 \ln n$ steps. I.e., once a 3-clause becomes a 2-clause, it is either satisfied or reduced to a 0-clause in $O(\ln n)$ steps.*

Proof: Say that t is a "busy time" if there exists at least one 2-clause or 1-clause at time t, and define a time-window $[r + 1,s]$ to be a "busy window" if time r is not busy but then each $t \in [r + 1,s]$ is a busy time. We will prove that for some constant c_2, with probability $1 - o(1)$, all busy windows have length at most $c_2 \ln n$.

Fix some r and s and consider the event that $[r+1,s]$ is a busy window. Since SC always decreases the total number of 1 and 2-clauses by one whenever it is positive, we must have generated at least $s-r$ new 2-clauses between r and s. Now, define an indicator variable for each 3-clause which has value one if the clause turns into a 2-clause between r and s. By Lemma 8.20 these variables are independent and the probability that a particular 3-clause turns into a 2-clause at a time t is at most $3/(2(n-2))$. Summing over t between r and s,

$$\text{Prob a 3-clause turns into a 2-clause during } [r, s]) \leq \frac{3(s-r)}{2(n-2)}.$$

Since there are cn clauses in all, the expected sum of the indicator variables is $cn\frac{3(s-r)}{2(n-2)} \approx \frac{3c(s-r)}{2}$. Note that $3c/2 < 1$, which implies the arrival rate into the queue of 2 and 1clauses is a constant strictly less than one. Using Chernoff bounds, if $s - r \geq c_2 \ln n$ for appropriate constant c_2, the probability that more than $s - r$ clauses turn into 2-clauses between r and s is at most $1/n^3$. Applying the union bound over all $O(n^2)$ possible choices of r and s, we get that the probability that any clause remains a 2 or 1-clause for more than $c_2 \ln n$ steps is $o(1)$.■

Now, assume the $1 - o(1)$ probability event of Lemma 8.21 that no clause remains a 2 or 1-clause for more than $c_2 \ln n$ steps. We will show that this implies it is unlikely the SC algorithm terminates in failure.

Suppose SC terminates in failure. This means that at some time t, the algorithm generates a 0-clause. At time $t - 1$, this clause must have been a 1-clause. Suppose the clause consists of the literal w. Since at time $t - 1$, there is at least one 1-clause, the shortest clause rule of SC selects a 1-clause and sets the literal in that clause to true. This other clause must have been $\bar{w}$. Let t_1 be the first time either of these two clauses, w or $\bar{w}$, became a 2-clause. We have $t - t_1 \leq c_2 \ln n$. Clearly, until time t, neither of these two clauses is picked by SC. So, the literals which are set to true during this period are chosen independent of these clauses. Say the two clauses were $w + x + y$ and $\bar{w} + u + v$ at the start. $x,y,u,$ and v must all be negations of literals set to true during steps t_1 to t. So, there are only $O\left((\ln n)^4\right)$ choices for x,y,u, and v for a given value of t. There are $O(n)$ choices of w, $O(n^2)$ choices of which two clauses i and j of the input become these w and $\bar{w}$, and n choices for t. Thus, there are $O\left(n^4(\ln n)^4\right)$ choices for what these clauses contain and which clauses they are in the input. On the other hand, for any given i and j, the probability that clauses i and j both match a given set of literals is $O(1/n^6)$. Thus the probability that these choices are actually realized is therefore $O\left(n^4(\ln n)^4/n^6\right) = o(1)$, as required.

8.8 Nonuniform Models of Random Graphs

So far we have considered the $G(n,p)$ random graph model in which all vertices have the same expected degree, and moreover degrees are concentrated close to their expectation. However, large graphs occurring in the real world tend to have *power law* degree distributions. For a power law degree distribution, the number $f(d)$ of vertices of degree d scales as $1/d^{\alpha}$ for some constant $\alpha > 0$.

One way to generate such graphs is to stipulate that there are $f(d)$ vertices of degree d and choose uniformly at random from the set of graphs with this degree distribution. Clearly, in this model the graph edges are not independent and this makes these random graphs harder to analyze. But the question of when phase transitions occur in random graphs with arbitrary degree distributions is still of interest. In this section, we consider when a random graph with a nonuniform degree distribution has a giant component. Our treatment in this section, and subsequent ones, will be more intuitive without providing rigorous proofs.

Consider a graph in which half of the vertices are degree one and half are degree two. If a vertex is selected at random, it is equally likely to be degree one or degree two. However, if we select an edge at random and walk to a random endpoint, the vertex is twice as likely to be degree two as degree one. In many graph algorithms, a vertex is reached by randomly selecting an edge and traversing the edge to reach an endpoint. In this case, the probability of reaching a degree i vertex is proportional to $i\lambda_i$ where λ_i is the fraction of vertices that are degree i.

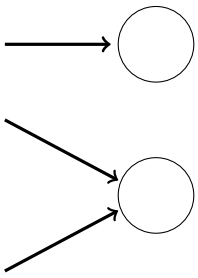


Figure 8.12: Probability of encountering a degree d vertex when following a path in a graph.

8.8.1 Giant Component in Graphs with Given Degree Distribution

Molloy and Reed address the issue of when a random graph with a nonuniform degree distribution has a giant component. Let λ_i be the fraction of vertices of degree i. There will be a giant component if and only if $\sum_{i=0}^{\infty} i(i-2)\lambda_i > 0$.

To see intuitively that this is the correct formula, consider exploring a component of a graph starting from a given seed vertex. Degree zero vertices do not occur except in the case where the vertex is the seed. If a degree one vertex is encountered, then that terminates the expansion along the edge into the vertex. Thus, we do not want to encounter too many degree one vertices. A degree two vertex is neutral in that the vertex is entered by one edge and left by the other. There is no net increase in the size of the

frontier. Vertices of degree i greater than two increase the frontier by $i - 2$ vertices. The vertex is entered by one of its edges and thus there are $i - 1$ edges to new vertices in the frontier for a net gain of $i - 2$. The $i\lambda_i$ in $(i - 2)i\lambda_i$ is proportional to the probability of reaching a degree i vertex and the $i - 2$ accounts for the increase or decrease in size of the frontier when a degree i vertex is reached.

Example: Consider applying the Molloy Reed conditions to the $G(n,p)$ model, and use p_i to denote the probability that a vertex has degree i, i.e., in analog to λ_i. It turns out that the summation $\sum_{i=0}^{n} i(i-2)p_i$ gives value zero precisely when $p = 1/n$, the point at which the phase transition occurs. At $p = 1/n$, the average degree of each vertex is one and there are $n/2$ edges. However, the actual degree distribution of the vertices is binomial, where the probability that a vertex is of degree i is given by $p_i = \binom{n}{i} p^i (1-p)^{n-i}$.

We now show $\lim_{n\to\infty} \sum_i i(i-2)p_i = 0 \text{ for } p_i = \binom{n}{i} p^i (1-p)^{n-i}$ that $\lim$ when $p = 1/n$.

$$\lim_{n\to\infty} \sum_{i=0}^{n} i(i-2) \binom{n}{i} \left(\frac{1}{n}\right)^i \left(1 - \frac{1}{n}\right)^{n-i}$$

$$= \lim_{n\to\infty} \sum_{i=0}^{n} i(i-2) \frac{n(n-1)\cdots(n-i+1)}{i!\, n^i} \left(1 - \frac{1}{n}\right)^{n} \left(1 - \frac{1}{n}\right)^{-i}$$

$$= \frac{1}{e} \lim_{n\to\infty} \sum_{i=0}^{n} i(i-2) \frac{n(n-1)\cdots(n-i+1)}{i!\, n^i} \left(\frac{n}{n-1}\right)^{i}$$

$$\leq \sum_{i=0}^{\infty} \frac{i(i-2)}{i!}.$$

To see that $\sum_{i=0}^{\infty} \frac{i(i-2)}{i!} = 0$, note that

$$\sum_{i=0}^{\infty} \frac{i}{i!} = \sum_{i=1}^{\infty} \frac{i}{i!} = \sum_{i=1}^{\infty} \frac{1}{(i-1)!} = \sum_{i=0}^{\infty} \frac{1}{i!}$$

and

$$\sum_{i=0}^{\infty} \frac{i^2}{i!} = \sum_{i=1}^{\infty} \frac{i}{(i-1)!} = \sum_{i=0}^{\infty} \frac{i+1}{i!} = \sum_{i=0}^{\infty} \frac{i}{i!} + \sum_{i=0}^{\infty} \frac{1}{i!} = 2\sum_{i=0}^{\infty} \frac{1}{i!}.$$

Thus,

$$\sum_{i=0}^{\infty} \frac{i(i-2)}{i!} = \sum_{i=0}^{\infty} \frac{i^2}{i!} - 2\sum_{i=0}^{\infty} \frac{i}{i!} = 0.$$

■

8.9 Growth Models

Many graphs that arise in the outside world started as small graphs that grew over time. In a model for such graphs, vertices and edges are added to the graph over time. In such a model there are many ways in which to select the vertices for attaching a new edge. One is to select two vertices uniformly at random from the set of existing vertices. Another is to select two vertices with probability proportional to their degree. This latter method is referred to as preferential attachment. A variant of this method would be to add a new vertex at each unit of time and with probability δ add an edge where one end of the edge is the new vertex and the other end is a vertex selected with probability proportional to its degree. The graph generated by this latter method is a tree with a power law degree distribution.

8.9.1 Growth Model Without Preferential Attachment

Consider a growth model for a random graph without preferential attachment. Start with zero vertices. At each unit of time a new vertex is created and with probability δ two vertices chosen at random are joined by an edge. The two vertices may already have an edge between them. In this case, we add another edge. So, the resulting structure is a multi-graph, rather then a graph. Since at time t, there are t vertices and in expectation only $O(\delta t)$ edges where there are t^2 pairs of vertices, it is very unlikely that there will be many multiple edges.

The degree distribution for this growth model is calculated as follows. The number of vertices of degree k at time t is a random variable. Let $d_k(t)$ be the expectation of the number of vertices of degree k at time t. The number of isolated vertices increases by one at each unit of time and decreases by the number of isolated vertices, $b(t)$, that are picked to be end points of the new edge. $b(t)$ can take on values 0,1, or 2. Taking expectations,

$$d_0(t+1) = d_0(t) + 1 - E(b(t)).$$

Now $b(t)$ is the sum of two 0-1 valued random variables whose values are the number of degree zero vertices picked for each end point of the new edge. Even though the two random variables are not independent, the expectation of $b(t)$ is the sum of the expectations of the two variables and is $2\delta\frac{d_0(t)}{t}$. Thus,

$$d_0(t+1) = d_0(t) + 1 - 2\delta\frac{d_0(t)}{t}.$$

The number of degree k vertices increases whenever a new edge is added to a degree $k-1$ vertex and decreases when a new edge is added to a degree k vertex. Reasoning as above,

$$d_k(t+1) = d_k(t) + 2\delta\frac{d_{k-1}(t)}{t} - 2\delta\frac{d_k(t)}{t}. \tag{8.4}$$

Note that this formula, as others in this section, is not quite precise. For example, the same vertex may be picked twice, so that the new edge is a self-loop. For $k << t$, this

problem contributes a minuscule error. Restricting k to be a fixed constant and letting $t \to \infty$ in this section avoids these problems.

Assume that the above equations are exactly valid. Clearly, $d_0(1) = 1$ and $d_1(1) = d_2(1) = \cdots = 0$. By induction on t, there is a unique solution to (8.4), since given $d_k(t)$ for all k, the equation determines $d_k(t+1)$ for all k. There is a solution of the form $d_k(t) = p_k t$, where p_k depends only on k and not on t, provided k is fixed and $t \to \infty$. Again, this is not precisely true since $d_1(1) = 0$ and $d_1(2) > 0$ clearly contradict the existence of a solution of the form $d_1(t) = p_1 t$.

Set $d_k(t) = p_k t$. Then,

$$(t+1)\,p_0 = p_0 t + 1 - 2\delta\frac{p_0 t}{t}$$

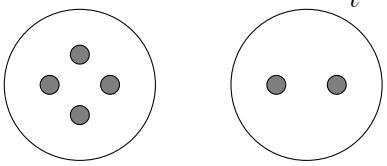


Figure 8.13: In selecting a component at random, each of the two components is equally likely to be selected. In selecting the component containing a random vertex, the larger component is twice as likely to be selected.

$$p_0 = 1 - 2\delta p_0$$

$$p_0 = \frac{1}{1+2\delta}$$

and

$$(t+1)\,p_k = p_k t + 2\delta\frac{p_{k-1}t}{t} - 2\delta\frac{p_k t}{t}$$

$$p_k = 2\delta p_{k-1} - 2\delta p_k$$

$$\begin{aligned} p_k &= \frac{2\delta}{1+2\delta}p_{k-1} \\ &= \left(\frac{2\delta}{1+2\delta}\right)^k p_0 \\ &= \frac{1}{1+2\delta}\left(\frac{2\delta}{1+2\delta}\right)^k. \end{aligned} \tag{8.5}$$

Thus, the model gives rise to a graph with a degree distribution that falls off exponentially fast with the degree.

The generating function for component size

Let $n_k(t)$ be the expected number of components of size k at time t. Then $n_k(t)$ is proportional to the probability that a randomly picked component is of size k. This is not the same as picking the component containing a randomly selected vertex (see Figure 8.13). Indeed, the probability that the size of the component containing a randomly selected vertex is k is proportional to $kn_k(t)$. We will show that there is a solution for $n_k(t)$ of the form $a_k t$ where a_k is a constant independent of t. After showing this, we focus on the generating function $g(x)$ for the numbers $ka_k(t)$ and use $g(x)$ to find the threshold for giant components.

Consider $n_1(t)$, the expected number of isolated vertices at time t. At each unit of time, an isolated vertex is added to the graph and an expected $\frac{2\delta n_1(t)}{t}$ many isolated vertices are chosen for attachment and thereby leave the set of isolated vertices. Thus,

$$n_1(t+1) = n_1(t) + 1 - 2\delta\frac{n_1(t)}{t}.$$

For $k > 1$, $n_k(t)$ increases when two smaller components whose sizes sum to k are joined by an edge and decreases when a vertex in a component of size k is chosen for attachment. The probability that a vertex selected at random will be in a size k component is $\frac{kn_k(t)}{t}$. Thus,

$$n_k(t+1) = n_k(t) + \delta\sum_{j=1}^{k-1}\frac{jn_j(t)}{t}\frac{(k-j)n_{k-j}(t)}{t} - 2\delta\frac{kn_k(t)}{t}.$$

To be precise, one needs to consider the actual number of components of various sizes, rather than the expected numbers. Also, if both vertices at the end of the edge are in the same k-vertex component, then $n_k(t)$ does not go down as claimed. These small inaccuracies can be ignored.

Consider solutions of the form $n_k(t) = a_k t$. Note that $n_k(t) = a_k t$ implies the number of vertices in a connected component of size k is $ka_k t$. Since the total number of vertices at time t is t, ka_k is the probability that a random vertex is in a connected component of size k. The recurrences here are valid only for k fixed as $t \to \infty$. So $\sum_{k=0}^{\infty} ka_k$ may be less than 1, in which case, there are nonfinite size components whose sizes are growing with t. Solving for a_k yields $a_1 = \frac{1}{1+2\delta}$ and $a_k = \frac{\delta}{1+2k\delta}\sum_{j=1}^{k-1} j(k-j)a_j a_{k-j}$.

Consider the generating function $g(x)$ for the distribution of component sizes where the coefficient of x^k is the probability that a vertex chosen at random is in a component of size k.

$$g(x) = \sum_{k=1}^{\infty} ka_k x^k.$$

Now, $g(1) = \sum_{k=0}^{\infty} k a_k$ is the probability that a randomly chosen vertex is in a finite sized component. For $\delta = 0$, this is clearly one, since all vertices are in components of size one. On the other hand, for $\delta = 1$, the vertex created at time one has expected degree log$n$ (since its expected degree increases by $2/t$ and $\sum_{t=1}^{n}(2/t) = \Theta(\log n)$); so, it is in a nonfinite size component. This implies that for $\delta = 1$, $g(1) < 1$ and there is a nonfinite size component. Assuming continuity, there is a $\delta_{critical}$ above which $g(1) < 1$. From the formula for the $a^0_i s$, we will derive the differential equation

$$g = -2\delta x g^0 + 2\delta x g g^0 + x$$

and then use the equation for g to determine the value of δ at which the phase transition for the appearance of a nonfinite sized component occurs.

Derivation of $g(x)$

From

$$a_1 = \frac{1}{1+2\delta}$$

and

$$a_k = \frac{\delta}{1+2k\delta} \sum_{j=1}^{k-1} j(k-j) a_j a_{k-j}$$

derive the equations

$$a_1 (1 + 2\delta) - 1 = 0$$

and

$$a_k (1 + 2k\delta) = \delta \sum_{j=1}^{k-1} j(k-j) a_j a_{k-j}$$

for $k \geq 2$. The generating function is formed by multiplying the k^{th} equation by kx^k and summing over all k. This gives

$$-x + \sum_{k=1}^{\infty} k a_k x^k + 2\delta x \sum_{k=1}^{\infty} a_k k^2 x^{k-1} = \delta \sum_{k=1}^{\infty} k x^k \sum_{j=1}^{k-1} j(k-j) a_j a_{k-j}.$$

Note that

$$g(x) = \sum_{k=1}^{\infty} k a_k x^k \text{ and } g^0(x) = \sum_{k=1}^{\infty} a_k k^2 x^{k-1}.$$

Thus,