



RWTHAACHEN
UNIVERSITY

Algorithmic Foundations of Data Science

Summer Semester 2023

Martin Grohe

Lehrstuhl Informatik 7 — Logic and the Theory of Discrete Systems

Introduction

Lectures

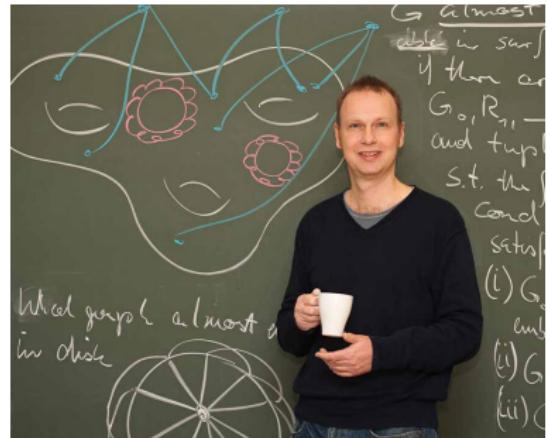
3-hour lecture will be given as
2 × 2-hours, but not every week.

Tuesday 16:30–18:00

Room: AH I

Thursday 8:30–10:00

Room: AH II



Overall, there will be 18 or 19 lectures. The exact dates (plus one backup date that we may not use) can be found in [RWTH Moodle](#).

Tutorials



Eva Fluck



Nina Runde

There will be weekly tutorials.

Friday 14:30–16:00

Room: AH II

First tutorial on Friday, April 28.

- ▶ There will be eight exercise sheets. Completing these successfully (at least 50% of possible points) is necessary for admittance to the examination.
- ▶ New exercise sheets will be released in Moodle on Mondays (exact dates can be found in Moodle). Each sheet has to be handed on Monday the week after in Moodle.
- ▶ Groups of up to three students are allowed (in fact, encouraged) to work together and hand in the solutions together.

You need to register your group in Moodle ("submission group assignment") until April 20, 10am.

- ▶ There will be written exams.
- ▶ First exam: Friday 11.08. 12:30-14:30
- ▶ Second exam: Tuesday 05.09. 15:00-17:00

Please check [RWTH Moodle](#) regularly. All information regarding the course can be found there, and new information will be released there as it becomes available.

Data science is about extracting knowledge and useable information from data.

The two main aspects are

Data Analysis and Systems Engineering

This is a course about the algorithmic foundations of data science. We will focus on the theoretical and mathematical aspects in the design and analysis of data science algorithms—it's a course in theoretical computer science.

Algorithms for Data Science

From the huge variety of algorithms that are needed in data science, we will focus on the following.

- ▶ Basic machine learning algorithms and the limits of learnability
- ▶ The curse of high dimensions and dimensionality reduction
- ▶ Random walks and the Markov-Chain-Monte-Carlo method
- ▶ Algorithms for massively parallel architectures (like MapReduce or Hadoop)
- ▶ Algorithms for streaming data

This is a class in theoretical computer science, and we will use **mathematical methods** (formal definitions and theorems, proofs, . . .).

Specifically, we will see techniques from:

- ▶ linear algebra, discrete mathematics, probability theory
- ▶ analysis of algorithms
- ▶ complexity theory
- ▶ information theory

Prerequisites

- ▶ Data Structures and Algorithms
- ▶ Database and Information Systems
- ▶ Computability and Complexity
- ▶ Mathematics: Discrete Structures, Linear Algebra, Probability Theory

Introduction

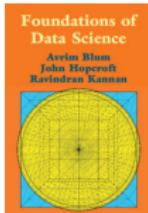
1. Understanding Machine Learning Algorithms
2. Information and Compression
3. Statistical Learning Theory
4. Multiplicative Weight Updates
5. High-Dimensional Data
6. Random Walks and Markov Chains
7. Algorithms for Massively Parallel Systems
8. Streaming Algorithms

Notation

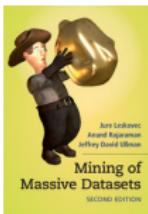
References

Textbooks

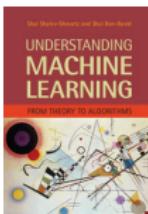
Much of the material covered in this course can be found in the following three textbooks. More detailed references will be given at the end of the chapters.



A. Blum, J. Hopcroft, R. Kannan. [Foundations of Data Science](#). Cambridge University Press, 2020.
Preliminary draft available at <https://www.cs.cornell.edu/jeh/book.pdf>.



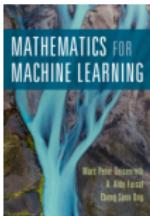
J. Leskovec, A. Rajaraman, J. Ullman. [Mining of Massive Datasets](#). Cambridge University Press 2014.
Available at <http://infolab.stanford.edu/~ullman/mmds/bookL.pdf>



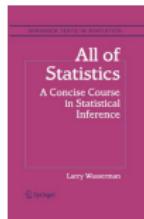
S. Shalev-Shwartz and S. Ben-David. [Understanding Machine Learning](#). Cambridge University Press, 2014. Available at <http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/index.html>

Textbooks (cont'd)

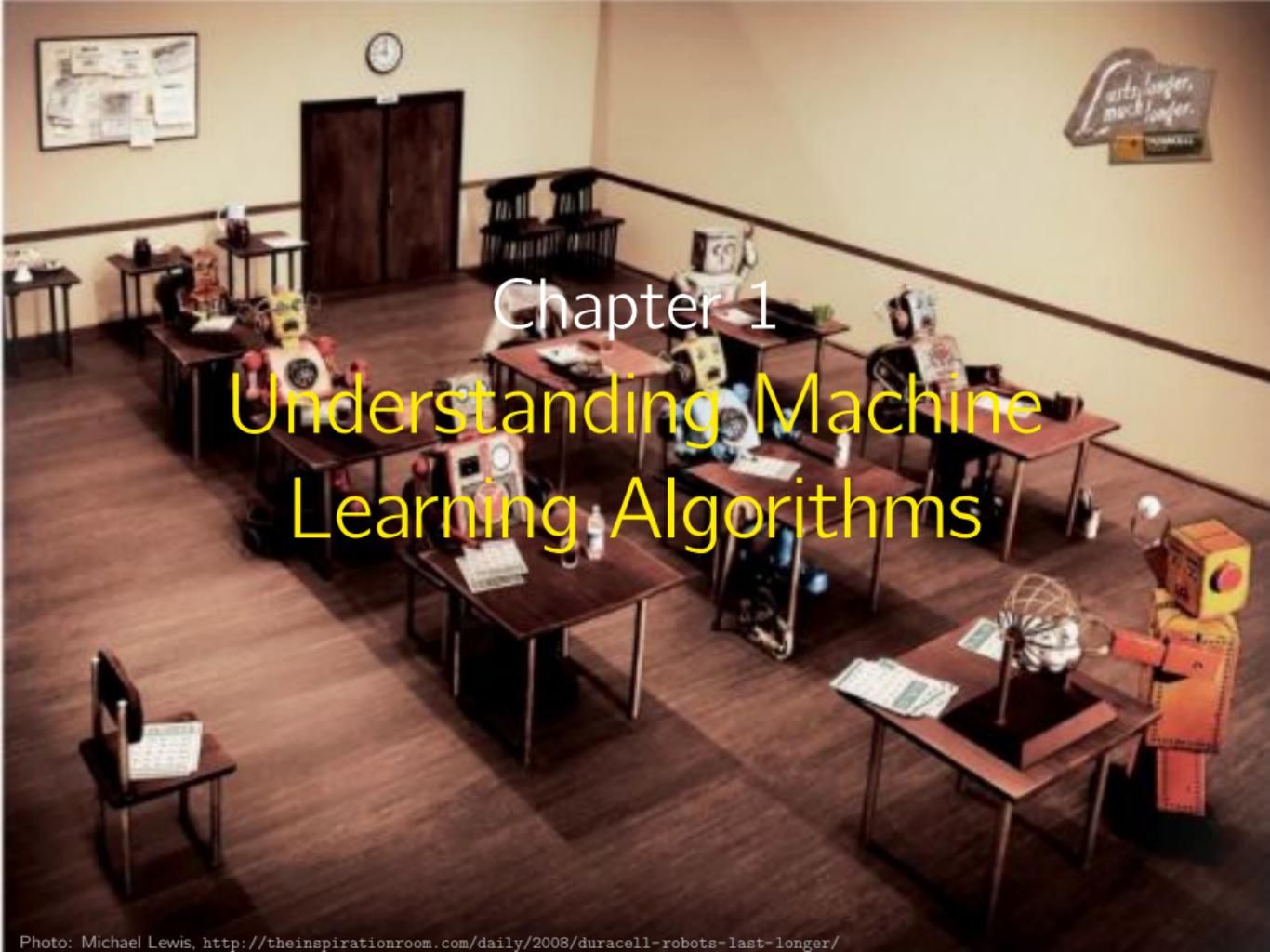
The following books contain useful mathematical background.



M.P. Deisenroth, A.A. Faisal, C.S. Ong. [Mathematics for Machine Learning](#). Cambridge University Press, 2020. Available at <https://mml-book.github.io/>



L. Wasserman. [All of Statistics](#). Springer, 2010.



A classroom interior featuring several wooden desks arranged in rows. On the desks are various educational displays, including a globe, a small robot, and a chalkboard sign that reads "DURACELL BATTERIES LAST LONGER". A large chalkboard sign hangs on the wall above the desks. The room has a warm, wooden floor and walls.

Chapter 1

Understanding Machine Learning Algorithms

Goals of this Chapter

- ▶ We will discuss a few basic machine learning algorithms, highlighting a number of ideas in data analysis.
- ▶ Our focus will be on different aspects of the theoretical analysis of such algorithms.

Disclaimer

This chapter is not meant as an introduction to machine learning.
Dedicated courses are offered for this.

1.1 Basic ML Concepts and Terminology

In machine learning, data is viewed as a collection of **data items**.

- ▶ A data item is represented by a so-called **feature vector**, a list (or vector) of properties, called **features** or **attributes**, that we can observe about the data item.
- ▶ The **selection** of suitable **features** is an important part of the modeling process, and it may strongly affect the quality of the results of the ML algorithms.
- ▶ Every feature has a **domain** of possible values.
- ▶ The Cartesian product of the domains, that is, the space of all possible feature vectors, is called the **instance space**.
- ▶ The number of features is the **dimension** of the instance space.

Data Model (cont'd)

- ▶ It is often convenient to assume that the domain of all features is the set \mathbb{R} of real numbers; non-numerical data are transformed into real numbers in some way.

For example, the Boolean values `true` and `false` may be represented by the reals `+1` and `-1` (or by `1` and `0`).

- ▶ Then the instance space is \mathbb{R}^ℓ , where ℓ is the dimension.
- ▶ If the instance space is \mathbb{R}^ℓ , then the whole dataset may be viewed as an $(n \times \ell)$ -matrix over the reals, where n is the number of data items.

Spam Detection

Data items are emails. Possible features are

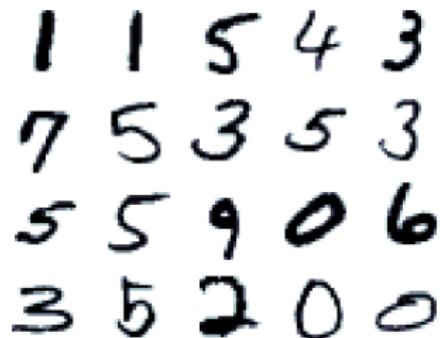
- ▶ **length** with domain \mathbb{N} (non-negative integers),
- ▶ **top-level domain of sender** with domain {de, uk, com, tv, ...},
- ▶ **occurrence of the word 'viagra'** with domain {*true*, *false*} (Boolean values).

Handwritten Digit Recognition

Data items are images of handwritten digits.

For example, the MNIST database has 20×20 pixel images with 1-byte greyscale values.

Features are the values at each pixel. That is, data items are represented as vectors in $\{0, \dots, 255\}^{400}$.



Sample from MNIST
database

Unsupervised learning

- ▶ try to detect patterns in data
- ▶ no explicit feedback supplied
- ▶ most important task is **clustering**

Reinforcement Learning

- ▶ try to find actions that maximise reward
- ▶ feedback is given as reward (or punishment)
- ▶ trial-and-error process

Forms of Learning (cont'd)

Supervised learning

- ▶ try to learn function from examples (input-output pairs)
- ▶ **classification** if function is finite-valued: try to predict values for future inputs
- ▶ **regression** if function is numerical: try to predict values for future inputs

Semi-supervised learning

- ▶ set-up like supervised learning, but only few and possibly faulty examples
- ▶ try to make the best out of the examples and data

The following two problems are typical classification problems.

Spam Detection

Data items are emails represented as described before. The task is to classify them into 'spam' and 'no spam'.

Because of the 2-valued outcome, which may be regarded as 'true' and 'false', we speak of a **Boolean classification problem**.

Handwritten Digit Recognition

Data items are handwritten digits represented as described before. The task is to classify them correctly as digits $\{0, \dots, 9\}$.

Note that this is a classification problem, but not a Boolean one.

Two Different Set-ups for Supervised Learning

Batch Learning

All examples are given at once, and the learner has to come up with a good hypothesis.

Online Learning

The examples are given one at a time, and the learner has to improve a hypothesis over time, learning from the new examples that come in.

In this course, we mostly focus on batch learning.

Active vs Passive Learning

Both active and passive learning are supervised learning scenarios.

Passive Learning

In a passive learning scenario, the training examples are just given to us, we cannot influence the selection of examples we see.

Active Learning

In an active learning scenario, we can actively choose specific data points and ask for their target value.

Both active learning and passive learning have their applications. In typical data science applications, we are in a passive learning scenario, so **we focus on passive learning in this course**.

Hypothesis Space

Suppose now we are in a supervised learning setting where we want to learn an unknown target function.

A learning algorithm chooses its hypothesis h from a pre-scribed hypothesis space \mathcal{H} .

Examples for hypothesis spaces

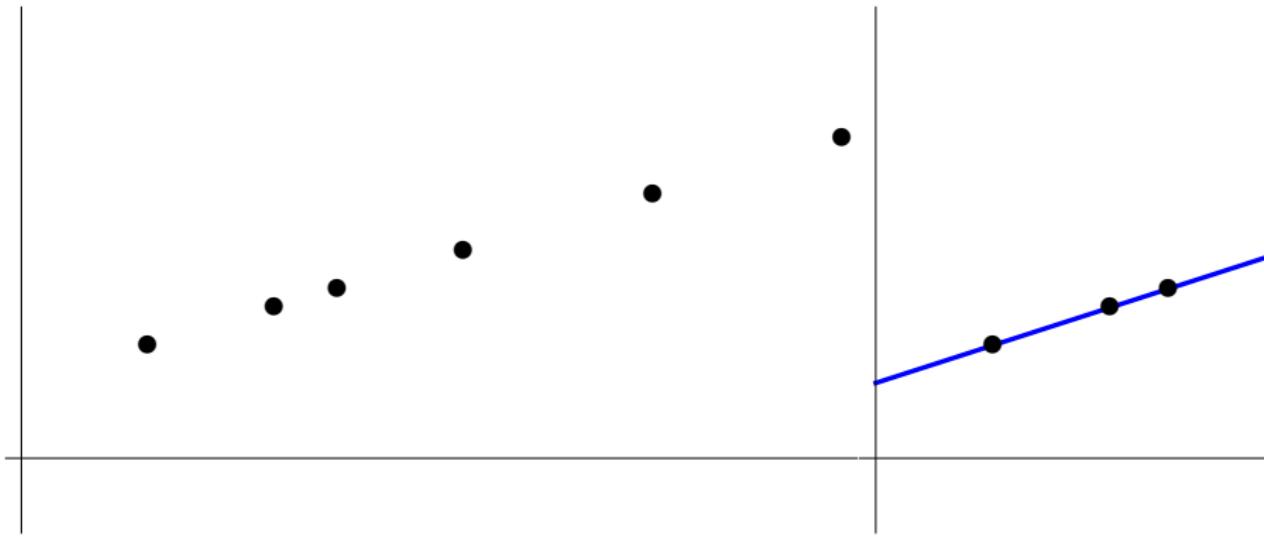
- ▶ all linear functions
- ▶ all polynomials, or all polynomials of a pre-scribed degree
- ▶ all functions that can be described by a decision tree (see Section 1.3)

Goal

The goal of the learning algorithm is to produce a hypothesis that generalises well, that is, approximates the target function well on all data points (and not only those in the training set).

- ▶ To evaluate how well a hypothesis generalises, we can evaluate it against some **test set**.
- ▶ In practice, we split our set of examples into a training sequence and a test set.
- ▶ It is important to keep the test set separate and to not use it to adapt the parameters; this would invalidate the test.

Example 1



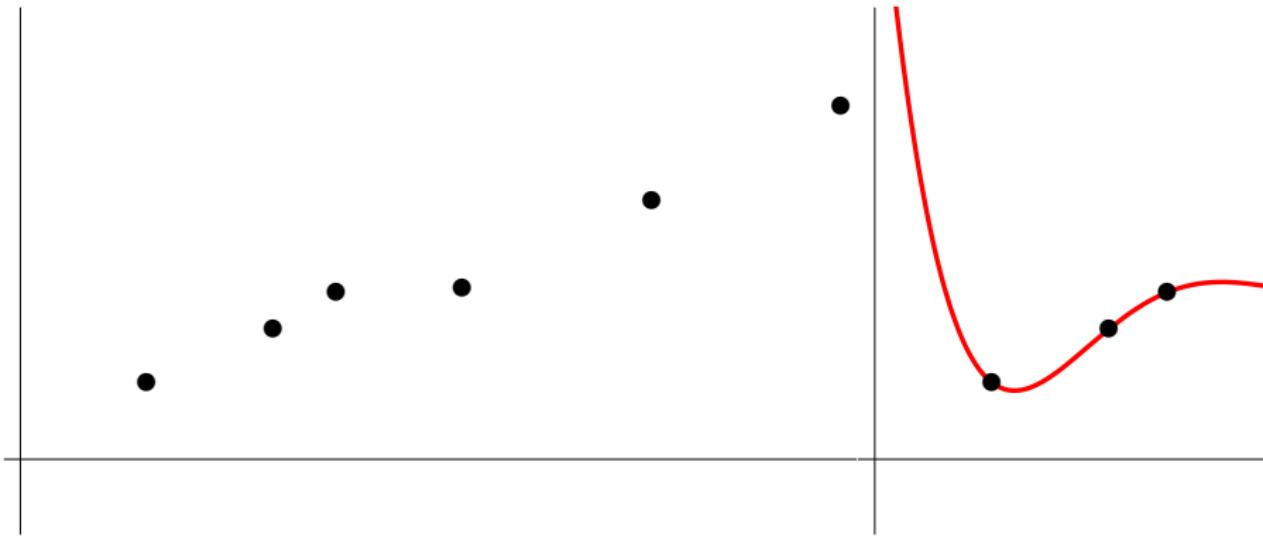
Empirical Observation

Simpler hypotheses tend to generalise better.

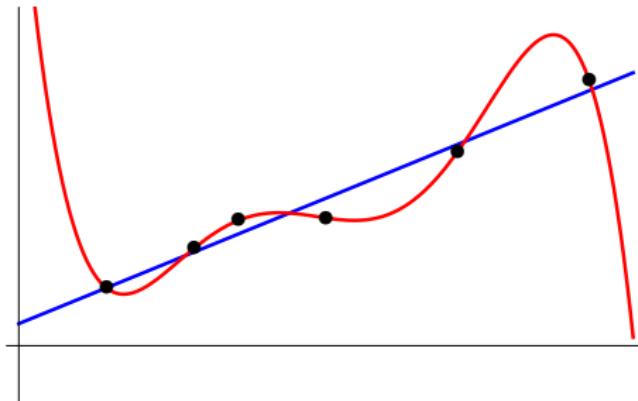
Occam's Razor

Choose the simplest hypothesis consistent with the data.

Example 2



Overfitting

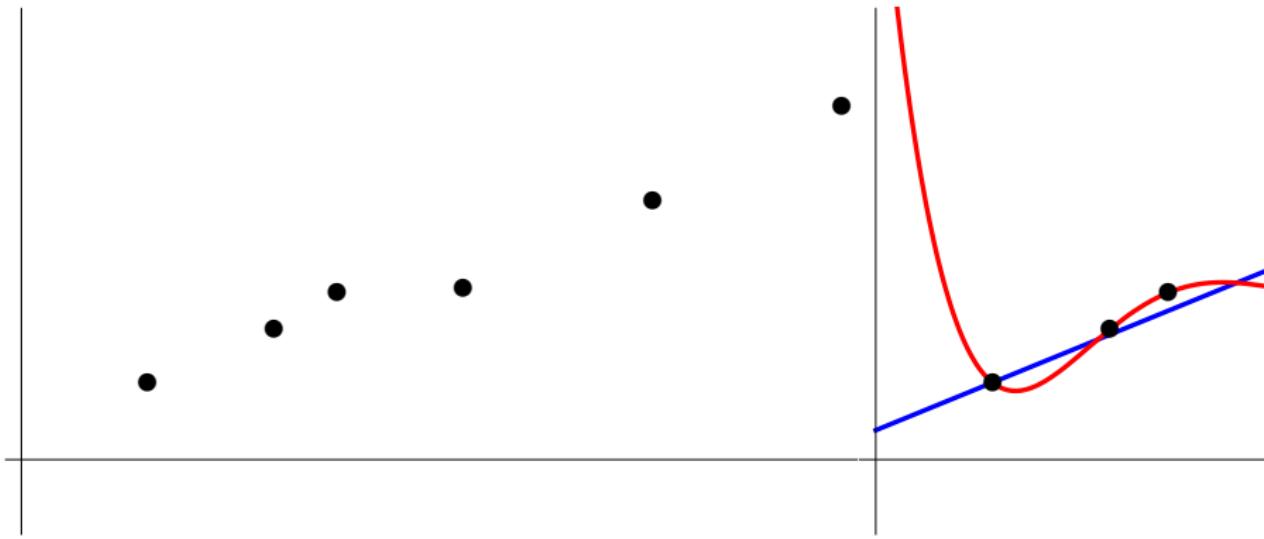


Trying to match the training examples exactly may lead to too complicated hypotheses that generalise badly.

This phenomenon is known as **overfitting**.

To avoid overfitting, it is often better to choose a simple hypothesis even if it doesn't match the data exactly.

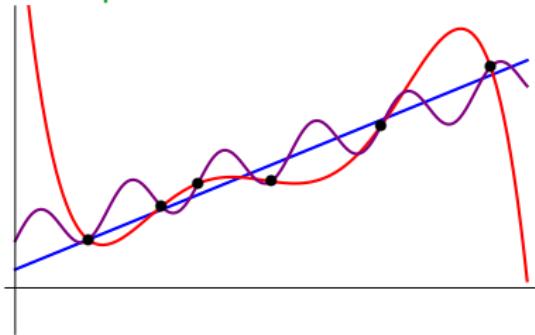
Example 2 (cont'd)



Realisable Learning Problems

A learning problem is **realisable** if the target function is in the hypothesis space.

Example 1.1



The target function may be the purple function, which is of the form $a \sin(x) + bx + c$.

- ▶ If the hypothesis space is the set of all polynomials, the learning problem is not realisable.
- ▶ If the hypothesis space is the set of all linear functions or polynomials in x and $\sin(x)$, the learning problem is realisable.

1.2 The Nearest Neighbour Algorithm

- ▶ The Nearest Neighbour algorithm may be the simplest learning algorithm.
- ▶ Yet in some situations, like handwritten digit recognition, it works fairly well.

Idea

- ▶ To predict the value of a function at a point x , we look at the known values of points close to the given one and assume that the value of x is similar.
- ▶ To avoid coincidences, it makes sense to look at several points close to x and then take the majority or average values.

Underlying Assumption

Data items that are close together have similar function values, or belong to the same class for a classification problem.

Background: Metric Spaces

A **metric** on a set \mathbb{X} is a function $d: \mathbb{X}^2 \rightarrow \mathbb{R}$ such that for all $x, y, z \in \mathbb{X}$:

Nonnegativity

$$d(x, y) \geq 0$$

and $d(x, y) = 0 \iff x = y;$

Symmetry

$$d(x, y) = d(y, x);$$

Triangle Inequality

$$d(x, z) \leq d(x, y) + d(y, z).$$

If d is a metric on \mathbb{X} , then the pair (\mathbb{X}, d) is a **metric space**.

Examples

- $\mathbb{X} = \mathbb{R}^\ell$ and d is the **Euclidean distance** (a.k.a. ℓ_2 -distance):

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{\ell} (x_i - y_i)^2}$$

- $\mathbb{X} = \mathbb{R}^\ell$ and d is the **Manhattan distance** (a.k.a. ℓ_1 -distance):

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{\ell} |x_i - y_i|$$

- $\mathbb{X} = \mathbb{R}^\ell$ and d is the **Chebychev distance** (a.k.a. **chess king distance** or ℓ_∞ -distance):

$$d(\mathbf{x}, \mathbf{y}) = \max_{1 \leq i \leq \ell} |x_i - y_i|$$

- $\mathbb{X} = \mathbb{D}_1 \times \mathbb{D}_2 \times \dots \times \mathbb{D}_\ell$ and d is the **Hamming distance**:

$$d(\mathbf{x}, \mathbf{y}) = |\{i \mid x_i \neq y_i\}|$$

Classification Problem

- ▶ We want to “learn” an unknown function f that associates a class $f(x) \in \mathbb{Y}$ with every data item $x \in \mathbb{X}$.
- ▶ More specifically, given “labelled examples”

$$(x_1, y_1), \dots (x_m, y_m),$$

where each $x_i \in \mathbb{X}$ is a data item and $y_i = f(x_i)$ the class it belongs to, our learning algorithm is supposed to produce a **classifier** that, given a data item x , predicts the value $f(x)$.

Nearest Neighbour Classifier

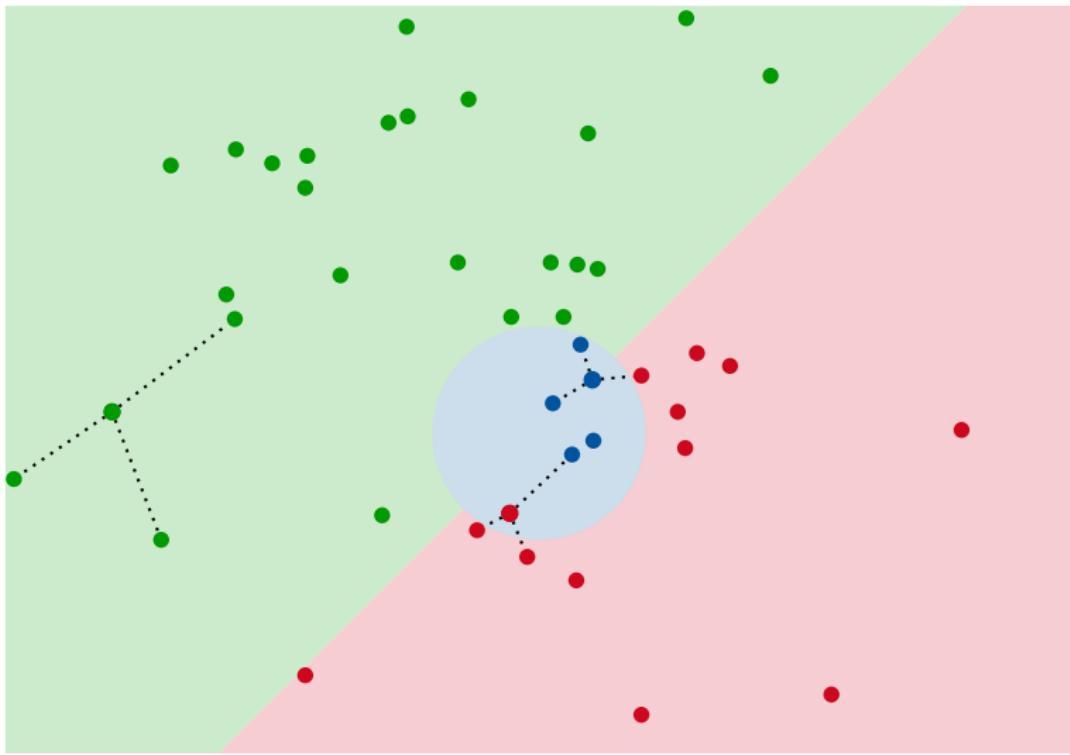
Classifier k -NEAREST NEIGHBOUR

- ▷ Based on labelled examples $(x_1, y_1), \dots (x_m, y_m)$
- ▷ k is some parameter

Input: $x \in \mathbb{X}$

1. Find the k nearest neighbours x_{i_1}, \dots, x_{i_k} of x in $\{x_1, \dots, x_m\}$.
2. Take a “majority vote”, that is, return the class y that appears most often among y_{i_1}, \dots, y_{i_k} (break ties arbitrarily).

Example



- ▶ Finding the nearest neighbours of a query point in a set of n data points requires time $O(n)$. For large data sets, this may be prohibitive.
- ▶ We can reduce the query time by preprocessing the data points into a suitable data structure.
- ▶ A typical data structure used for nearest neighbour search is a ***k-d-tree***, a version of binary search trees for k -dimensional data.
- ▶ It may be enough to use data structures supporting approximate nearest neighbour searches, for example, using **locality sensitive hashing**.

How Many Neighbours?

Question

How do we choose the parameter k for the k -Nearest Neighbour classifier?

- ▶ If we choose $k = 1$, then our hypothesis (= classifier) is guaranteed to be consistent with the examples, but we tend to overfit.
- ▶ The larger we choose k , the simpler the hypothesis gets, because we average out irregularities. But at some point (maybe $k \geq 10$) we start to over-simplify.
- ▶ The best value of k depends on the application (and can be viewed as a parameter that should be “learned” as well).

1.3 Learning Decision Trees

A **decision tree** represents a finite-valued function on feature vectors.

Syntax

A decision tree is a rooted tree with labelled nodes and edges.

- ▶ Every internal node of the tree is labelled by an (input) feature.
- ▶ Every edge is labeled by a value or range of values for the feature labelling its source node.
- ▶ Every leaf is labeled with an output value.

The edge labelling must be such that every value for a feature labelling a node appears at exactly one edge going out of that node.

Semantics

The function value for an input vector \mathbf{x} is the value at the leaf of the unique path in the tree whose edges are labelled by the feature values in \mathbf{x} .

Decision Trees for Classification

We only consider decision trees for functions that have

- ▶ finite-valued features,
(numerical inputs can be partitioned into finitely many intervals)
- ▶ finitely many output values (often just two: 'yes'/'no' or
'true'/'false' or '1/0')

That is, we are dealing with classification problems (**Boolean classification** in the case of just two outputs).

Remark 1.2

Decision trees can also be used for regression, but we won't consider this here.

Example

Classification Problem

Decide when to attend class.

Input Features

Feature	Values
Weather	Sunny (☀), Cloudy (☁), Rainy (🌧), Snow (❄)
Day of the week	Mon, Tue, Wed, Thu, Fri
Party the night before	yes (🥳), no (😴)
Topic	DB, ML, Alg, Logic
Start time	8–10, 11–15, 16–20
Lecturer	Codd ( , Karp ( , Rabin ( , Valiant (img alt="Portrait of Valiant" data-bbox="926 834 956 891/>)

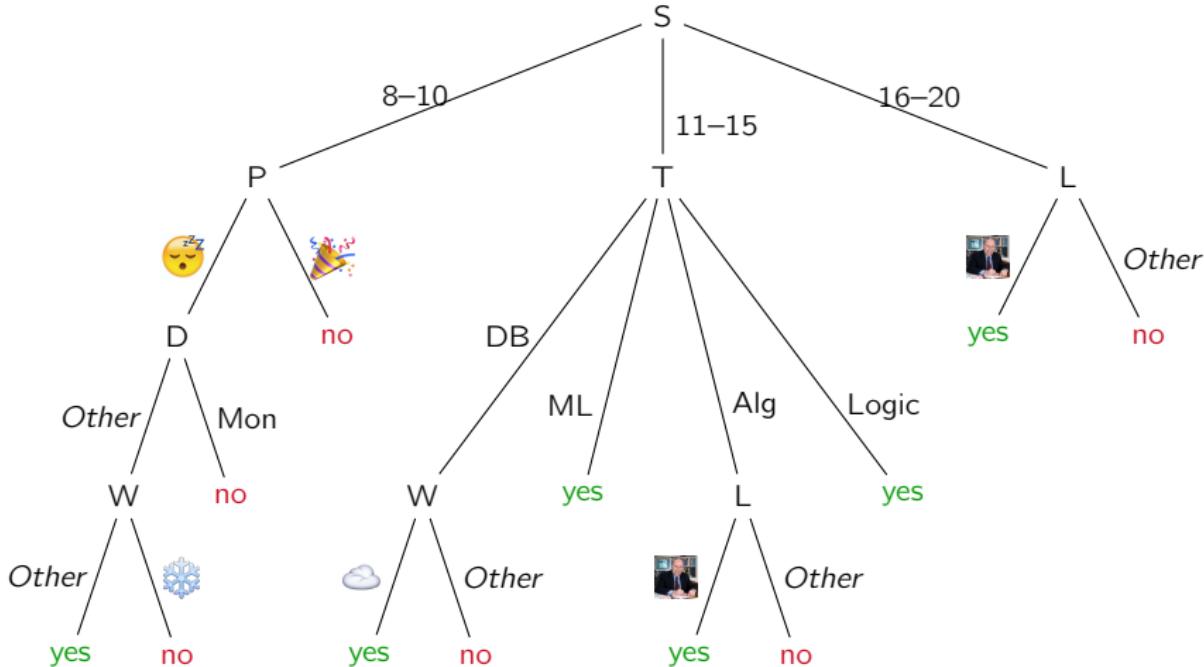
Example (cont'd)

Data (Training Sequence)

Example	W	D	P	T	S	L	Output
x ₁		Mon		DB	8–10		y ₁ = no
x ₂		Wed		ML	11–15		y ₂ = yes
x ₃		Thu		Alg	11–15		y ₃ = no
x ₄		Wed		DB	11–15		y ₄ = yes
x ₅		Thu		Logic	8–10		y ₅ = no
x ₆		Tue		Logic	16–20		y ₆ = yes
x ₇		Tue		Alg	16–20		y ₇ = no
x ₈		Wed		Logic	11–15		y ₈ = yes
x ₉		Thu		DB	11–15		y ₉ = no
x ₁₀		Thu		DB	8–10		y ₁₀ = yes
x ₁₁		Wed		ML	16–20		y ₁₁ = no
x ₁₂		Thu		DB	16–20		y ₁₂ = no
x ₁₃		Tue		Alg	11–15		y ₁₃ = yes
x ₁₄		Tue		Alg	8–10		y ₁₄ = no
x ₁₅		Mon		Alg	8–10		y ₁₅ = no
x ₁₆		Tue		Alg	8–10		y ₁₆ = yes

Example (cont'd)

“True” decision tree:



Greedy Algorithm for Building Decision Trees

Input: Set \mathcal{A} of features, set S of examples

Objective: Compute decision tree t .

1. if $S = \emptyset$ then
2. create leaf t with arbitrary value
 ▷ e.g., majority value for parent node
3. else if all examples in S have the same y -value then
4. create leaf t with that y -value
5. else
6. choose feature $A \in \mathcal{A}$ that discriminates best
 between examples in S
7. create new node t with feature A
8. partition examples in S according to their A -value
 into parts S_1, \dots, S_m
9. recursively call algorithm on $\mathcal{A} \setminus \{A\}$ and the S_i
 and attach resulting trees t_i as children to t
10. return t

- ▶ There are different heuristics for picking the “best” feature A .
One is to pick the feature that yields the highest **information gain** (in a precise sense defined in terms of information theory).
- ▶ The algorithm tends to build too complicated decision trees that overfit the data.
In practice, trees are pruned with various heuristics.

Decision Trees for Boolean Functions

We can also represent Boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ by decision trees.

- ▶ Features are represented by the variables; we always take variables X_1, \dots, X_n for an n -ary Boolean function.
- ▶ Feature values are $\{0, 1\}$, so all inner nodes of a decision tree for a Boolean function have exactly two children

The decision tree learning problem for Boolean formulas is to construct a decision tree representing an unknown Boolean function from a few examples of the form

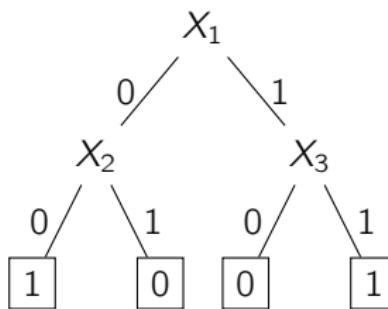
$$(\underbrace{(x_1, x_2, \dots, x_n)}_{\text{values for the variables } X_i}, \underbrace{y}_{\text{target value}}).$$

Example

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Consider the Boolean function
 $f : \{0, 1\}^3 \rightarrow \{0, 1\}$ described by the
following table:

Decision tree:



Functions with Large Decision Trees

Theorem 1.3

- (1) Every Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is represented by a decision tree height n with $2^{n+1} - 1$ nodes.
- (2) There is a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ such that every decision tree representing f has height at least n and at least $2^{n+1} - 1$ nodes.

Proof of Theorem 1.3.

- (1) $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be represented by a full binary tree where each variable appears exactly once on every path from the root to a leaf. The height of this tree is n . A full binary tree of height n has 2^n leaves and $2^n - 1$ inner nodes. Thus it has $2^{n+1} - 1$ nodes overall.
- (2) Take the parity function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ defined by

$$f(x_1, \dots, x_n) = \begin{cases} 1 & \text{if the number of } x_i \text{ that are 1 is even,} \\ 0 & \text{otherwise.} \end{cases}$$

Let D be a decision tree for f .

Then on every path in D , all variables must appear, because for every setting of $n - 1$ variables the value of the n th variable determines the value of the parity function.

Thus every root-to-leaf path in D has length n , and as each inner node has two children, D is a full binary tree of height n . □

Complexity of Computing Decision Trees

Theorem 1.4

Computing a smallest decision tree for a given set of examples is NP-hard.

More precisely:

The following decision problem is NP-complete.

DECISION TREE

Instance Examples

$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in \{0, 1\}^n \times \{0, 1\}$ for a Boolean function in n variables, integer $k \geq 0$

Problem Decide if there is a decision tree with at most k nodes that is consistent with the examples

Lemma 1.5

A binary tree has k leaves if and only if it has $2k - 1$ nodes.

Proof.

Induction on k .

□

Recall that a **vertex cover** of a graph G is a set $X \subseteq V(G)$ such that every edge of G has at least one endvertex in X .

The following problem is well-known to be NP-complete.

VERTEX COVER

Instance Graph G , integer $k \geq 0$

Problem Decide if G has a vertex cover of size at most k

Proof of Theorem 1.4.

We reduce VERTEX COVER to DECISION TREE.

Let G be a graph with vertex set $V(G) = [n]$ and edge set $E(G) = \{e_1, \dots, e_m\}$, where $e_i = v_i w_i$.

We construct a list of $m + 1$ examples

$$(\mathbf{x}_0, y_0), \dots, (\mathbf{x}_m, y_m) \in \{0, 1\}^n \times \{0, 1\},$$

where $\mathbf{x}_i = (x_{i1}, \dots, x_{in})$, as follows:

- ▶ $x_{0j} = 0$ for all j , and $y_0 = 0$;
- ▶ $x_{iv_i} = 1$ and $x_{iw_i} = 1$ and $x_{jj} = 0$ for $j \in [n] \setminus \{v_i, w_i\}$ and $y_i = 1$.

Here the variables X_1, \dots, X_n of the Boolean function we are looking for, correspond to the vertices of the graph.

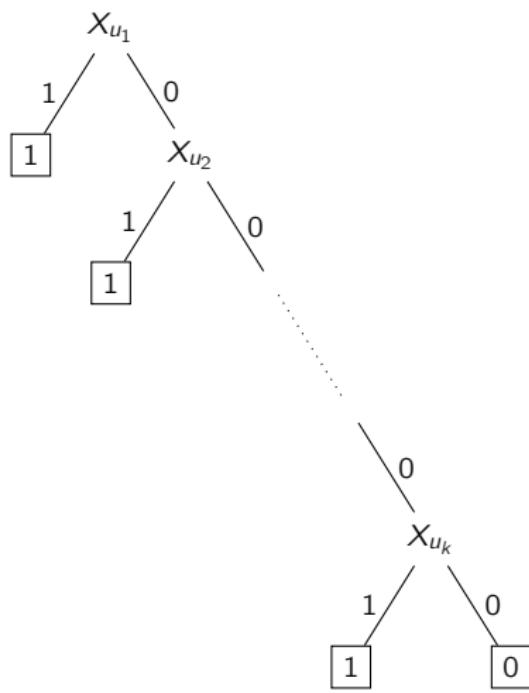
Claim

G has a vertex cover of size at most k iff there is a decision tree with at most $(k + 1)$ leaves that is consistent with the examples.

Clearly, the claim implies the theorem.

Proof of the claim.

If $\{u_1, \dots, u_k\}$ is a vertex cover of G , then



is a decision tree consistent with the examples. To see this, let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be the function computed by this decision tree. Then

$$f(\mathbf{x}_0) = f(0, \dots, 0) = 0 = y_0.$$

Moreover, for $i \in [m]$ there is a $j \in [k]$ such that u_j is an endvertex of the edge e_i , that is, $u_j = v_i$ or $u_j = w_i$. Hence $x_{iu_j} = 1$, and this implies

$$f(\mathbf{x}_i) = 1 = y_i.$$

Now suppose that D is a decision tree with at most $k + 1$ leaves that is consistent with the examples, and let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be the function represented by D . Let t_1, \dots, t_ℓ be the “1”-leaves of D . Then $\ell \leq k$, and for every t_j at least one variable X_u on the path from the root to t_j is set to true, because

$$f(0, \dots, 0) = f(\mathbf{x}_0) = 0.$$

Let X_{u_j} be the first such variable.

Observe that for every edge $e_i = v_i w_i$ there is a leaf t_j such that on the path from the root to t_j no variable other than X_{v_i}, X_{w_i} is set to true, because $f(\mathbf{x}_i) = y_i = 1$.

Thus $\{u_1, \dots, u_\ell\}$ is a vertex cover of G . □

1.4 Linear Classifiers

Background from Linear Algebra: Scalar Product

The **scalar product** (or **dot product**) of two vectors
 $\mathbf{x} = (x_1, \dots, x_\ell)^\top, \mathbf{y} = (y_1, \dots, y_\ell)^\top \in \mathbb{R}^\ell$ is the real number

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{\ell} x_i y_i.$$

Background: Euclidean Norm and the Cauchy-Schwarz Inequality

The **Euclidean norm** of a vector $\mathbf{x} \in \mathbb{R}^\ell$ is the nonnegative real number

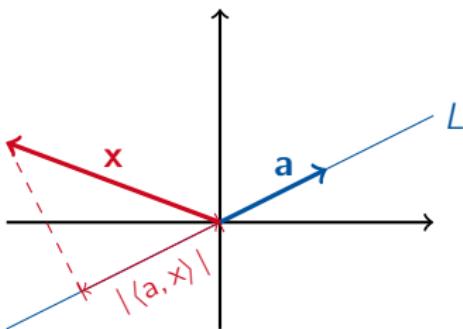
$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\sum_{i=1}^{\ell} x_i^2}.$$

Cauchy-Schwarz Inequality

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \|\mathbf{y}\|.$$

Background: Geometric Meaning of the Scalar Product

Let $\mathbf{a} \in \mathbb{R}^\ell$ such that $\|\mathbf{a}\| = 1$, and let L be the line through \mathbf{a} .



Let $\mathbf{x} \in \mathbb{R}^\ell$ be another vector. Then

- ▶ $|\langle \mathbf{a}, \mathbf{x} \rangle|$ is the length of the projection of \mathbf{x} into L ;
- ▶ the sign of $\langle \mathbf{a}, \mathbf{x} \rangle$ indicates whether the projection of \mathbf{x} into L points into the same direction as \mathbf{a} (positive) or into the opposite direction (negative).

If \mathbf{a} is not a unit vector, then $\frac{|\langle \mathbf{a}, \mathbf{x} \rangle|}{\|\mathbf{a}\|}$ is the length of the projection of \mathbf{x} into L .

Background: Hyperplanes

- ▶ An (affine) hyperplane in \mathbb{R}^ℓ is an affine subspace of dimension $\ell - 1$.

Examples: a line in \mathbb{R}^2 , a plane in \mathbb{R}^3

- ▶ A hyperplane can be described as the set of all solutions $(x_1, \dots, x_\ell) \in \mathbb{R}^\ell$ to an equation

$$a_1x_1 + \dots + a_\ell x_\ell = b,$$

where $a_1, \dots, a_\ell, b \in \mathbb{R}$.

Short: $\langle \mathbf{a}, \mathbf{x} \rangle = b$ or $\langle \mathbf{a}, \mathbf{x} \rangle - b = 0$.

- ▶ A hyperplane $P = \{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x} \rangle = b\}$ is **homogeneous** if $b = 0$, or equivalently, if $\mathbf{0} \in P$, that is, the hyperplane goes through the origin.

Background: Halfspaces

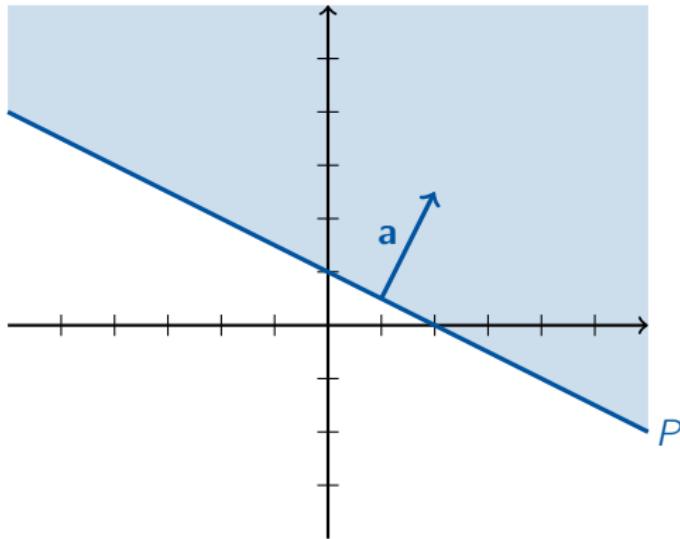
- ▶ A **halfspace** in \mathbb{R}^ℓ is the set of all points on one side of a hyperplane.
- ▶ Formally, a halfspace can be described as the set of all solutions \mathbf{x} to an inequality

$$\langle \mathbf{a}, \mathbf{x} \rangle - b \geq 0,$$

where $\mathbf{a} \in \mathbb{R}^\ell$ and $b \in \mathbb{R}$.

- ▶ A halfspace $H = \{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x} \rangle - b \geq 0\}$ is **homogeneous** if $b = 0$.

Example



Hyperplane $P = \{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x} \rangle = b\}$ for $\mathbf{a} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ and $b = 1$.

Halfspace $H = \{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x} \rangle \geq b\}$

Linear Classification

- ▶ We consider a Boolean classification problem with instance space; the goal is to learn an unknown target function $f : \mathbb{R}^\ell \rightarrow \{+1, -1\}$.
- ▶ The input of our learning algorithm is a sequence

$$S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)) \in \mathbb{R}^\ell \times \{+1, -1\}$$

- ▶ The hypothesis space consists of **linear separators**, that is, functions $h : \mathbb{R}^\ell \rightarrow \mathbb{R}$ of the form

$$h(\mathbf{x}) = \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle - b) = \begin{cases} +1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle - b > 0, \\ 0 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle - b = 0, \\ -1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle - b < 0 \end{cases}$$

for some $\mathbf{w} \in \mathbb{R}^\ell$ (called **weight vector**) and $b \in \mathbb{R}$ (called **bias**).

Realisability and Consistent Hypotheses

- ▶ Note that formally a linear classification problem is not realisable, because the target function has range $\{+1, -1\}$ and all hypotheses have range $\{+1, 0, -1\}$.
- ▶ However, if the target function f is of the form

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle - b \geq 0, \\ -1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle - b < 0 \end{cases}$$

then for every set of finitely many points $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^\ell$ we can find a hypothesis h of the form $h(\mathbf{x}) = \text{sgn}(\langle \mathbf{w}', \mathbf{x} \rangle - b')$ such that $f(\mathbf{x}_i) = h(\mathbf{x}_i)$.

- ▶ We say that a hypothesis h is **consistent** with a training sequence $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m))$ if $h(\mathbf{x}_i) = y_i$ for all $i \in [m]$.

Reduction to the Homogeneous Case

A **homogeneous linear separator** is a function of the form
 $\mathbf{x} \mapsto \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle)$.

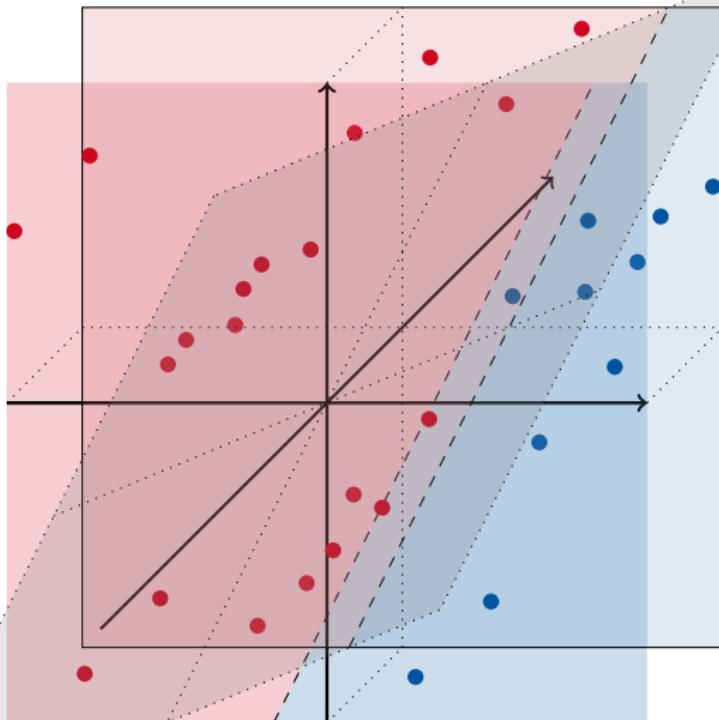
Lemma 1.6

Let $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)) \in (\mathbb{R}^\ell \times \{-1, 1\})^m$ with
 $\mathbf{x}_i = (x_{i1}, \dots, x_{i\ell}) \in \mathbb{R}^\ell$ and $\mathbf{w} = (w_1, \dots, w_\ell) \in \mathbb{R}^\ell$, $b \in \mathbb{R}$. Then the following are equivalent.

- (i) $\mathbf{x} \mapsto \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle - b)$ is a linear separator consistent with S .
- (ii) $\widehat{\mathbf{x}} \mapsto \text{sgn}(\langle \widehat{\mathbf{w}}, \widehat{\mathbf{x}} \rangle)$ is a homogeneous linear separator consistent with $\widehat{S} = ((\widehat{\mathbf{x}}_1, y_1), \dots, (\widehat{\mathbf{x}}_m, y_m))$, where $\widehat{\mathbf{x}}_i := (x_{i1}, \dots, x_{i\ell}, 1)$ and $\widehat{\mathbf{w}} = (w_1, \dots, w_\ell, -b)$.

Thus it suffices to design algorithms finding homogeneous linear separators.

Example



Empirical Risk Minimisation (ERM)

In the following, our goal will be to solve the following algorithmic problem:

Instance: Training data $S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)) \in (\mathbb{R}^\ell \times \{-1, 1\})^m$

Problem: Compute weight vector $\mathbf{w} \in \mathbb{R}^\ell$ such that the homogeneous linear separator $\mathbf{x} \mapsto \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle)$ minimises the **training error**, that is, the number of i such that $\text{sgn}(\langle \mathbf{w}, \mathbf{x}_i \rangle) \neq y_i$

Remarks 1.7

- ▶ Learning by finding a hypothesis minimising the training error is known as **empirical risk minimisation**. It is not clear that such a hypothesis also minimises the generalisation error (cf. Chapter 3).
- ▶ We will mostly focus on the **realisable case** where we assume that a homogeneous linear separator consistent with S exists.

Background: Linear Programming

Linear programs are optimisation problems that aim to minimise (or maximise) a linear function over real vectors subject to linear inequalities.

LP

Instance Matrix $A \in \mathbb{R}^{m \times n}$, vectors $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$

Problem Compute $\mathbf{x} \in \mathbb{R}^n$

$$\begin{aligned} & \text{minimising} && \langle \mathbf{c}, \mathbf{x} \rangle \\ & \text{subject to} && A\mathbf{x} \geq \mathbf{b}. \end{aligned}$$

Theorem 1.8 (Khachiyan 1979)

LP can be solved in polynomial time (polynomial in m, n and the bit-size of the coefficients).

- ▶ In the following, we always let

$$S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)) \in (\mathbb{R}^\ell \times \{-1, 1\})^m,$$

where $\mathbf{x}_i = (x_{i1}, \dots, x_{i\ell})^\top$.

- ▶ Even though formally it is a tuple, it will often be convenient to think of S as a set and use notations like $(\mathbf{x}, y) \in S$.

Linear Inequalities for a Separator

Observation 1.9

For all $\mathbf{w}, \mathbf{x} \in \mathbb{R}^\ell$ and $y \in \{-1, 1\}$,

$$\operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle) = y \iff y \langle \mathbf{w}, \mathbf{x} \rangle > 0.$$

Lemma 1.10

The following are equivalent:

- (i) There is a $\mathbf{w} \in \mathbb{R}^\ell$ such that the homogeneous linear separator $\mathbf{x} \mapsto \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle)$ is consistent with S , that is, $\operatorname{sgn}(\langle \mathbf{w}, \mathbf{x}_i \rangle) = y_i$ for all $i \in [m]$.
- (ii) There is a $\mathbf{w} \in \mathbb{R}^\ell$ such that $y_i \cdot \langle \mathbf{w}, \mathbf{x}_i \rangle > 0$ for all $i \in [m]$.
- (iii) There is a $\mathbf{w} \in \mathbb{R}^\ell$ such that $y_i \cdot \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1$ for all $i \in [m]$.
- (iv) There is a $\mathbf{w} \in \mathbb{R}^\ell$ such that $A_S \mathbf{w} \geq \mathbf{1}$, where $A_S = (a_{ij})_{i \in [m], j \in [\ell]} \in \mathbb{R}^{m \times \ell}$ is the matrix with entries $a_{ij} = y_i \cdot x_{ij}$ and $\mathbf{1} = (\underbrace{1, \dots, 1}_{m \text{ times}})^\top$.

Proof of Lemma 1.10.

(i) \iff (ii): follows immediately from Observation 1.9.

(ii) \implies (iii): Assume $y_i \cdot \langle \mathbf{w}, \mathbf{x}_i \rangle > 0$ for all $i \in [m]$ and let

$$\varepsilon := \min_{i \in [m]} y_i \cdot \langle \mathbf{w}, \mathbf{x}_i \rangle .$$

Then $\varepsilon > 0$. Let $\mathbf{w}' := \frac{1}{\varepsilon} \mathbf{w}$. Then $y_i \cdot \langle \mathbf{w}', \mathbf{x}_i \rangle \geq 1$ for all $i \in [m]$.

(iii) \implies (ii): is trivial.

(iii) \iff (iv): is immediate from the definitions of the scalar product and vector-matrix product. □

ERM by Linear Programming

Corollary 1.11

Assume that there is homogeneous linear separator consistent with S . Then we can find such a linear separator by solving the following linear program:

$$\begin{aligned} \min_{\mathbf{w} \in \mathbb{R}^\ell} \quad & \langle \mathbf{0}, \mathbf{w} \rangle \\ \text{subject to} \quad & A_S \mathbf{w} \geq \mathbf{1}, \end{aligned}$$

where $\mathbf{0} = \underbrace{(0, \dots, 0)}_{\ell \text{ times}}^\top$ and A_S and $\mathbf{1}$ are defined as on the previous slide.

The Perceptron Algorithm

Algorithm PERCEPTRON

Input: Training sequence S such that there is homogeneous linear separator consistent with S .

Objective: Compute $\mathbf{w} \in \mathbb{R}^\ell$ such that $\mathbf{x} \mapsto \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle)$ is consistent with S .

1. $\mathbf{w} \leftarrow \mathbf{0}$
2. repeat
3. for all $(\mathbf{x}, y) \in S$ do
4. if $\text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle) \neq y$ then
5. $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$
6. until $\text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle) = y$ for all $(\mathbf{x}, y) \in S$

Remarks 1.12

- ▶ The Perceptron algorithm is one of the earliest machine learning algorithms.
- ▶ It can be viewed as a precursor of neural networks.

Perceptron in Hardware

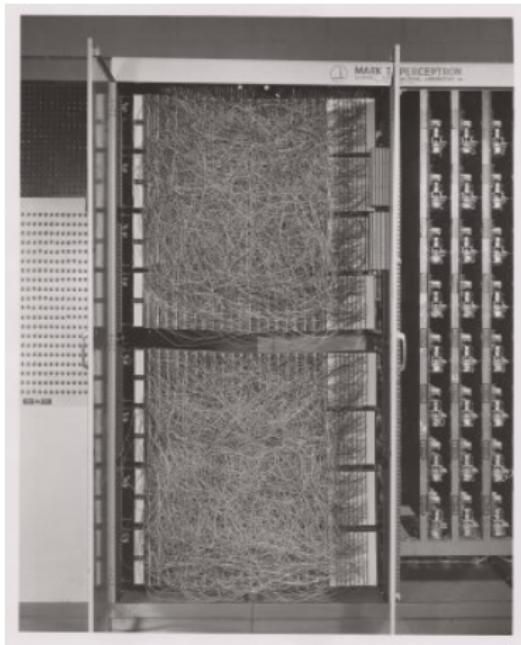
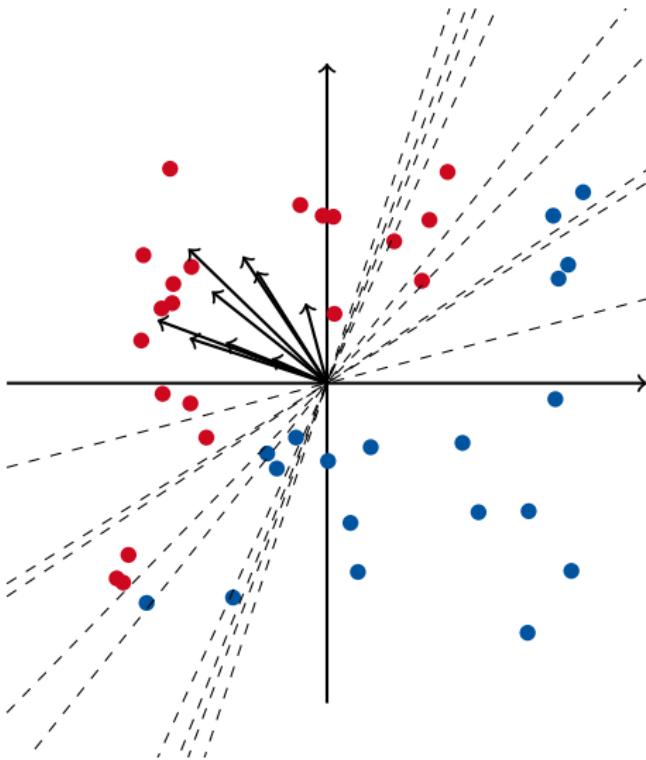


Photo: Wikipedia, <https://en.wikipedia.org/wiki/Perceptron>

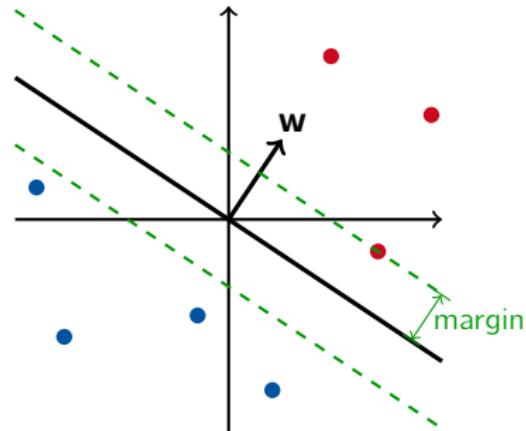
Mark I Perceptron Machine (for image recognition)

Example



Suppose that $h : \mathbf{x} \mapsto \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle)$ is a linear separator consistent with a sequence S of examples. The **margin** of h with respect to S is

$$\min_{(\mathbf{x}, y) \in S} \frac{|\langle \mathbf{w}, \mathbf{x} \rangle|}{\|\mathbf{w}\|}.$$



Theorem 1.13

Suppose that there is a homogeneous linear separator consistent with S of margin γ . Let

$$\lambda := \max_{(\mathbf{x}, y) \in S} \|\mathbf{x}\|.$$

Then the perceptron algorithm applied to S finds a linear separator after at most $(\lambda/\gamma)^2$ updates of \mathbf{w} .

Proof of the Theorem.

Suppose that $\mathbf{w}^* \in \mathbb{R}^\ell$ such that $\mathbf{x} \mapsto \text{sgn}(\langle \mathbf{w}^*, \mathbf{x} \rangle)$ is consistent with S of margin $\geq \gamma$.

W.l.o.g.: $\|\mathbf{w}^*\| = 1$.

Let $\mathbf{w}_0 = \mathbf{0}$ and \mathbf{w}_i the value of \mathbf{w} after the i -th update in line 5 of the PERCEPTRON algorithm. Let $k \in \mathbb{N} \cup \{\infty\}$ be the number of updates the algorithm makes on input S .

For all $i \in \mathbb{N}$, let $(\mathbf{x}_i, y_i) \in S$ with $\text{sgn}(\langle \mathbf{w}_i, \mathbf{x}_i \rangle) \neq y_i$ be the example used for updating \mathbf{w}_i in line 5 of the algorithm during the $(i+1)$ st iteration of the loop. Then

$$\mathbf{w}_{i+1} = \mathbf{w}_i + y_i \mathbf{x}_i. \quad (*)$$

Note that $\text{sgn}(\langle \mathbf{w}_i, \mathbf{x}_i \rangle) \neq y_i$ is equivalent to

$$y_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle \leq 0. \quad (**)$$

Goal: Prove that $k \leq \left(\frac{\lambda}{\gamma}\right)^2$.

Claim 1

For all $i < k$,

$$\langle \mathbf{w}_{i+1}, \mathbf{w}^* \rangle - \langle \mathbf{w}_i, \mathbf{w}^* \rangle \geq \gamma.$$

Proof: We have $\text{sgn}(\langle \mathbf{w}^*, \mathbf{x}_i \rangle) = y_i$ and $|\langle \mathbf{w}^*, \mathbf{x}_i \rangle| \geq \gamma$, which implies $y_i \langle \mathbf{w}^*, \mathbf{x}_i \rangle \geq \gamma$. Thus

$$\begin{aligned}\langle \mathbf{w}_{i+1}, \mathbf{w}^* \rangle - \langle \mathbf{w}_i, \mathbf{w}^* \rangle &= \langle \mathbf{w}_i + y_i \mathbf{x}_i - \mathbf{w}_i, \mathbf{w}^* \rangle && \text{by } (\star) \\ &= y_i \langle \mathbf{w}^*, \mathbf{x}_i \rangle \geq \gamma.\end{aligned}$$

□

Claim 2

For all $i < k$,

$$\|\mathbf{w}_{i+1}\|^2 - \|\mathbf{w}_i\|^2 \leq \lambda^2.$$

Proof: We have

$$\begin{aligned}\|\mathbf{w}_{i+1}\|^2 &= \langle \mathbf{w}_i + y_i \mathbf{x}_i, \mathbf{w}_i + y_i \mathbf{x}_i \rangle && \text{by } (\star) \\ &= \|\mathbf{w}_i\|^2 + 2y_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle + \|\mathbf{x}_i\|^2 && \text{since } y_i^2 = 1\end{aligned}$$

$$\leq \|\mathbf{w}_i\|^2 + \|\mathbf{x}_i\|^2 \quad \text{by } (\star)$$

$$\leq \|\mathbf{w}_i\|^2 + \lambda^2 \quad \text{because } \|\mathbf{x}_i\| \leq \lambda.$$

□

Claim 1 $\implies \langle \mathbf{w}_k, \mathbf{w}^* \rangle \geq \gamma k.$

Claim 2 $\implies \|\mathbf{w}_k\|^2 \leq \lambda^2 k.$

Cauchy-Schwarz Inequality \implies

$$\langle \mathbf{w}_k, \mathbf{w}^* \rangle \leq \|\mathbf{w}_k\| \cdot \|\mathbf{w}^*\| = \|\mathbf{w}_k\|.$$

Thus $\gamma^2 k^2 \leq \lambda^2 k$. This implies that $k \leq \left(\frac{\lambda}{\gamma}\right)^2$, that is, the algorithm stops after at most $\left(\frac{\lambda}{\gamma}\right)^2$ updates. □

Support Vector Machines

The Perceptron algorithm and the Linear Programming approach always find linear separators if they exist, and they do so efficiently. However, the separators found are only consistent, but not optimal in any sense.

An algorithm known as **support vector machine** finds linear separators of maximum margin. Computing such a separators amounts to solving the following quadratic optimisation problem.

SVM

Instance $S \in (\mathbb{R}^\ell \times \{-1, 1\})^m$

Problem Compute $\mathbf{w} \in \mathbb{R}^\ell$

minimising $\|\mathbf{w}\|^2$

subject to $y \langle \mathbf{w}, \mathbf{x} \rangle \geq 1 \quad \text{for all } (\mathbf{x}, y) \in S$

Remark 1.14

Note that SVM has the same linear constraints as the linear program we used for finding a separator, but we add a quadratic optimisation target.

SVMs Maximise the Margin

Theorem 1.15

Suppose that there is a homogeneous linear separator consistent with S .

Let $\mathbf{w}^* \in \mathbb{R}^\ell$ be an optimal solution to SVM. Then

$h^* : \mathbf{x} \mapsto \text{sgn}(\langle \mathbf{w}^*, \mathbf{x} \rangle)$ is a homogeneous linear separator consistent with S .

Furthermore, for every homogeneous linear separator

$h : \mathbf{x} \mapsto \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle)$ consistent with S , the margin of h w.r.t. S is smaller than or equal to the margin of h^* w.r.t. S .

Remark 1.16

As opposed to linear programs, quadratic programs such as SVM can no longer be solved in polynomial time. However, in practice SVM can be solved reasonably well using nonlinear (gradient descent) optimisation techniques.

Proof of Theorem 1.15.

Since $y \langle \mathbf{w}^*, \mathbf{x} \rangle \geq 1 > 0$ for all $(\mathbf{x}, y) \in S$, we have $h^*(\mathbf{x}) = \text{sgn}(\langle \mathbf{w}^*, \mathbf{x} \rangle) = y$ for all $(\mathbf{x}, y) \in S$. Thus h^* is consistent with S .

Let

$$\gamma^* := \min_{(\mathbf{x}, y) \in S} \frac{|\langle \mathbf{w}^*, \mathbf{x} \rangle|}{\|\mathbf{w}^*\|}$$

be the margin of h^* with respect to S . To prove the minimality of γ^* , let $\mathbf{w} \in \mathbb{R}^\ell$ such that $h : \mathbf{x} \mapsto \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle)$ consistent with S . Then $y \langle \mathbf{w}, \mathbf{x} \rangle > 0$ for all $(\mathbf{x}, y) \in S$. Let

$$\varepsilon := \min_{(\mathbf{x}, y) \in S} y \langle \mathbf{w}, \mathbf{x} \rangle$$

and

$$\mathbf{w}' := \frac{1}{\varepsilon} \mathbf{w}.$$

Then $y \langle \mathbf{w}', \mathbf{x} \rangle \geq 1$ for all $(\mathbf{x}, y) \in S$. Thus \mathbf{w}' satisfies all constraints of SVM, and therefore by the minimality of \mathbf{w}^* we have $\|\mathbf{w}^*\| \leq \|\mathbf{w}'\|$.

Let γ be the margin of h w.r.t. S . Since $\min_{(x,y) \in S} y \langle \mathbf{w}, \mathbf{x} \rangle = \varepsilon$, we have

$$\gamma = \min_{(x,y) \in S} \frac{|\langle \mathbf{w}, \mathbf{x} \rangle|}{\|\mathbf{w}\|} = \min_{(x,y) \in S} \frac{y \langle \mathbf{w}, \mathbf{x} \rangle}{\|\mathbf{w}\|} = \frac{\varepsilon}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}'\|}.$$

Thus

$$\gamma = \frac{1}{\|\mathbf{w}'\|} \leq \frac{1}{\|\mathbf{w}^*\|} \leq \min_{(x,y) \in S} \frac{|\langle \mathbf{w}^*, \mathbf{x} \rangle|}{\|\mathbf{w}^*\|} = \gamma^*.$$



The Non-Realisable Case

- ▶ The methods for finding linear separators that we considered so far all assume that there is a (homogeneous) linear separator consistent with the training data S .
- ▶ In practice, we often don't have such a clean cut separation, and we need to find a linear separator that separates the data points reasonably well.
- ▶ There is a “soft” version of SVMs that allows some inconsistencies, but penalises them depending on how far an example lies on the wrong side of the separating hyperplane.
- ▶ In the following, we will consider another classic “soft” linear separation method known as **logistic regression**.

Logistic Regression

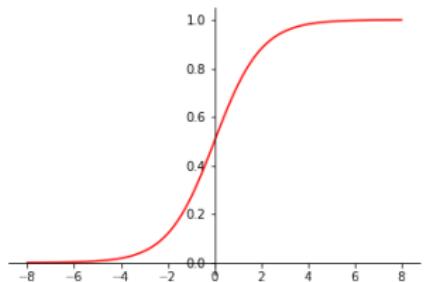
The idea of logistic regression is to not only construct a linear separator, but also a score between 0 and 1 indicating how confident we are that a point \mathbf{x} has answer is +1.

Logistic regression applied to training data $S \in (\mathbb{R}^\ell \times \{-1, 1\})^m$ returns a function $f : \mathbb{R}^\ell \rightarrow [0, 1]$ of the form

$$f_w(\mathbf{x}) = \text{sig}(\langle \mathbf{w}, \mathbf{x} \rangle)$$

for some $\mathbf{w} \in \mathbb{R}^\ell$, where $\text{sig} : \mathbb{R} \rightarrow [0, 1]$ is the **sigmoid function** (a.k.a. **logistic function**)

$$\text{sig}(z) := \frac{1}{1 + e^{-z}}.$$



Remark 1.17

This is the homogeneous version of logistic regression. We can also include a bias parameter and define

$$f_w(\mathbf{x}) = \text{sig}(\langle \mathbf{w}, \mathbf{x} \rangle - b).$$

Semantics of Logistic Regression

$$f_w(\mathbf{x}) = \text{sig} (\langle \mathbf{w}, \mathbf{x} \rangle)$$

expresses the confidence that the correct classification value for \mathbf{x} is 1:

- ▶ if $\langle \mathbf{w}, \mathbf{x} \rangle$ is large, then \mathbf{x} is far on the '+' side of the hyperplane defined by \mathbf{w} , and $f_w(\mathbf{x})$ is close to 1 (high confidence that correct class is 1);
- ▶ if $-\langle \mathbf{w}, \mathbf{x} \rangle$ is large, then \mathbf{x} is far on the '-' side of the hyperplane defined by \mathbf{w} , and $f_w(\mathbf{x})$ is close to 0 (high confidence that correct class is -1);
- ▶ if $|\langle \mathbf{w}, \mathbf{x} \rangle|$ is small, then \mathbf{x} is close to the hyperplane defined by \mathbf{w} , and $f_w(\mathbf{x})$ is halfway between 0 and 1 (low confidence).

Training a Logistic Regression Function

- ▶ To find the optimal parameters \mathbf{w} for our training data S , we want to minimise the training error, that is, maximise the confidence of giving the correct answer for all $(\mathbf{x}, y) \in S$.
- ▶ Thus we want to maximize

$$\begin{cases} f_w(\mathbf{x}) & \text{if } y = 1, \\ 1 - f_w(\mathbf{x}) & \text{if } y = -1. \end{cases}$$

simultaneously for all $(\mathbf{x}, y) \in S$.

- ▶ Note that $\text{sig}(z) = 1 - \text{sig}(-z)$ and therefore

$$f_w(y\mathbf{x}) = \text{sig}(y \langle \mathbf{w}, \mathbf{x} \rangle) = \begin{cases} f_w(\mathbf{x}) & \text{if } y = 1, \\ 1 - f_w(\mathbf{x}) & \text{if } y = -1. \end{cases}$$

- ▶ Overall, we maximize

$$\prod_{(\mathbf{x}, y) \in S} \text{sig}(y \langle \mathbf{w}, \mathbf{x} \rangle).$$

Logistic Regression as Minimisation

We want to find a $\mathbf{w} \in \mathbb{R}^\ell$ maximising $\prod_{(x,y) \in S} \text{sig}(y \langle \mathbf{w}, x \rangle)$.

Instead of maximising this product, it is numerically more convenient to maximise its logarithm (a sum) or minimise its negative logarithm:

$$-\ln \left(\prod_{(x,y) \in S} \text{sig}(y \langle \mathbf{w}, x \rangle) \right) = \sum_{(x,y) \in S} \ln (1 + \exp(-y \langle \mathbf{w}, x \rangle)).$$

LOGISTIC REGRESSION

Instance $S \in (\mathbb{R}^\ell \times \{-1, 1\})^m$

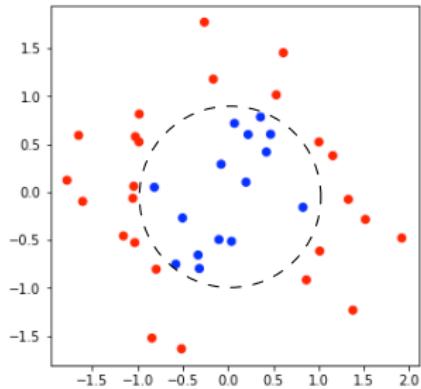
Problem Compute $\mathbf{w} \in \mathbb{R}^\ell$ minimising
 $\sum_{(x,y) \in S} \ln (1 + \exp(-y \langle \mathbf{w}, x \rangle))$.

This turns out to be a **convex minimisation problem** that can be solved efficiently using gradient descent.

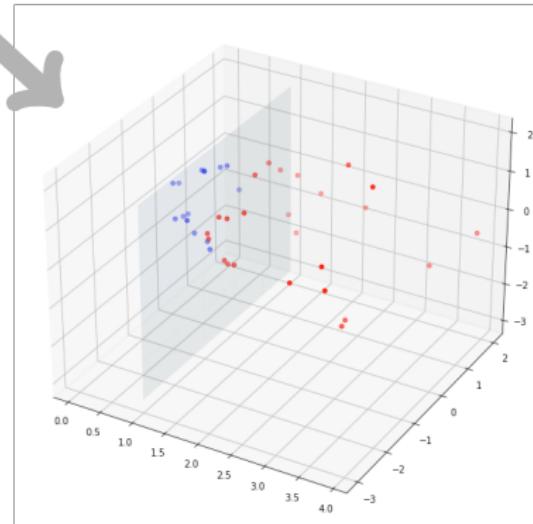
Linear Separation for Nonlinear Problems

- ▶ Learning just linear separators may seem very restrictive.
- ▶ However, linear separation methods can also be used to learn nonlinear separators.
- ▶ The trick is to map the data points to a higher dimensional space; such a mapping may transform nonlinear functions on the original instance space to linear functions in the new higher-dimensional space.

Example



$$(x, y) \mapsto (x^2 + y^2, x, y)$$



Quadratic Separators

Suppose our instance space is \mathbb{R}^2 and the hypothesis space consists of the **quadratic separators** of the form

$$(x_1, x_2) \mapsto \text{sgn}(w_1 x_1^2 + w_2 x_1 x_2 + w_3 x_2^2 + w_4 x_1 + w_5 x_2 + w_6).$$

To learn the parameters w_1, \dots, w_6 , we define a transformation $\tau : \mathbb{R}^2 \rightarrow \mathbb{R}^5$ by

$$\tau(x_1, x_2) = (x_1^2, x_1 x_2, x_2^2, x_1, x_2).$$

Then quadratic separators for a training sequence

$$S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)) \in (\mathbb{R}^2 \times \{+1, -1\})^m$$

correspond to linear separators for the transformed training sequence

$$\tau(S) := ((\tau(\mathbf{x}_1), y_1), \dots, (\tau(\mathbf{x}_m), y_m)) \in (\mathbb{R}^5 \times \{+1, -1\})^m.$$

1.5 k-Means Clustering

Task

Put a collection of data points into k clusters. (Here the number k of clusters is fixed in advance.)

We leave it open what exactly a **cluster** is and appeal to an intuitive understanding of the term: points in the same cluster should be close together; the distance between clusters should be large.

Thus again we need a metric on our instance space. We assume that the instance space is \mathbb{R}^ℓ and use the Euclidean metric.

This is an example of an **unsupervised learning** problem, in fact the only example we shall see in this course.

Centroid Based Clustering

- ▶ In the centroid based clustering, each of the clusters is represented by a point \mathbf{z} , which we take to be the **centroid** (a.k.a. “centre of mass” or “mean”), that is, the point that minimises the squared distance to all points in the cluster.
- ▶ Each data point \mathbf{x} will be associated with the cluster whose centroid is closest to it.

Formally, the problem can be stated as follows:

Centroid Clustering

Instance Points $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^\ell$, number $k \in \mathbb{N}$

Problem Find points $\mathbf{z}^1, \dots, \mathbf{z}^k \in \mathbb{R}^\ell$ and a partition C^1, \dots, C^k of $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ that minimise

$$\sum_{j=1}^k \sum_{x \in C^j} \|\mathbf{x} - \mathbf{z}^j\|^2$$

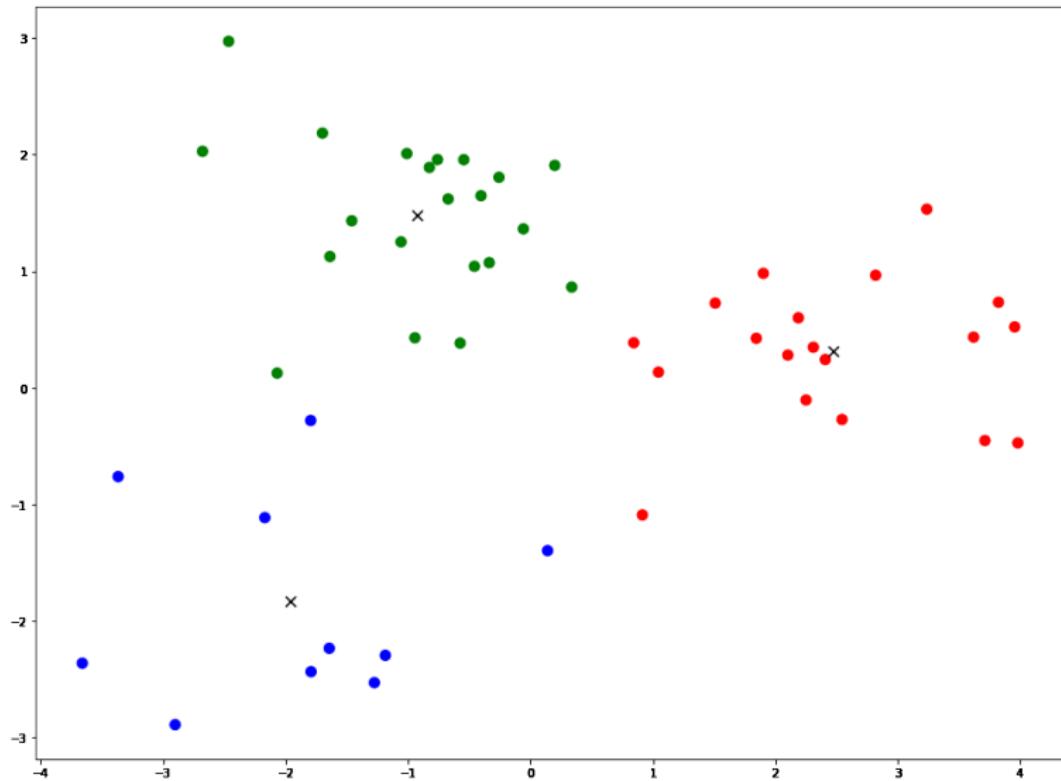
The k-Means Clustering Algorithm

Algorithm K-MEANS

Input: $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^\ell$, $k \in \mathbb{N}$

1. Choose initial “centroids” $\mathbf{z}^1, \dots, \mathbf{z}^k$ (for example, randomly)
2. repeat
3. $C^j \leftarrow \emptyset$ for all $j \in [k]$
4. for $i \leftarrow 1$ to n do
5. $j \leftarrow \arg \min_j \|\mathbf{x}_i - \mathbf{z}^j\|$
 (If there is a tie, choose the smallest j)
6. add \mathbf{x}_i to C^j
7. $\mathbf{z}^j \leftarrow \frac{\sum_{x \in C^j} \mathbf{x}}{|C^j|}$ for all $j \in [k]$
8. until C^1, \dots, C^k no longer change
9. return C^1, \dots, C^k

Example



Theorem 1.18

The k-Means algorithm always halts in a finite number of steps.

Lemma

Let $C \subseteq \mathbb{R}^\ell$ be a finite nonempty set, and let $\mathbf{z} := \frac{1}{|C|} \sum_{\mathbf{x} \in C} \mathbf{x}$.

Then for all $\mathbf{y} \in \mathbb{R}^\ell$ with $\mathbf{y} \neq \mathbf{z}$,

$$\sum_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{z}\|^2 < \sum_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{y}\|^2.$$

Proof.

Let $\mathbf{y} \in \mathbb{R}^\ell$ with $\mathbf{y} \neq \mathbf{z}$. Then

$$\begin{aligned} \sum_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{y}\|^2 &= \sum_{\mathbf{x} \in C} \|(\mathbf{x} - \mathbf{z}) + (\mathbf{z} - \mathbf{y})\|^2 \\ &= \sum_{\mathbf{x} \in C} \left(\|\mathbf{x} - \mathbf{z}\|^2 + 2 \langle \mathbf{x} - \mathbf{z}, \mathbf{z} - \mathbf{y} \rangle + \|\mathbf{z} - \mathbf{y}\|^2 \right) \\ &> \sum_{\mathbf{x} \in C} \left(\|\mathbf{x} - \mathbf{z}\|^2 + 2 \langle \mathbf{x} - \mathbf{z}, \mathbf{z} - \mathbf{y} \rangle \right) \end{aligned} \quad \text{because } \mathbf{y} \neq \mathbf{z}$$

$$\begin{aligned}
 &= \sum_{x \in C} \|x - z\|^2 + 2 \left\langle z - y, \sum_{x \in C} (x - z) \right\rangle \\
 &= \sum_{x \in C} \|x - z\|^2,
 \end{aligned}$$

because

$$\sum_{x \in C} (x - z) = \sum_{x \in C} x - |C|z = 0. \quad \square$$

Proof of the Theorem.

If the algorithm does not stop, then the same clustering must appear twice during the execution. We shall prove that this can never happen.

We need some preparation: For $s = 1, 2, \dots$, we let

- ▶ z_s^j be the j -th centroid,
- ▶ C_s^j be the j -th cluster

after the main loop (lines 3–7) has been executed the s -th time. Furthermore, we let

- ▶ $c_s(i)$ be the unique j such that $x_i \in C_s^j$.

We define the **total distance** and the **index sum** after round s to be

$$\Delta_s := \sum_{j=1}^k \sum_{x \in C_s^j} \|x - z_s^j\|^2,$$

$$\iota_s := \sum_{i=1}^n c_s(i).$$

Suppose for contradiction that the algorithm does not stop.

Claim

For every $s \geq 1$, either $\Delta_{s+1} < \Delta_s$ or $\Delta_{s+1} = \Delta_s$ and $\iota_{s+1} < \iota_s$.

Proof: If the algorithm does not stop, the clustering cannot remain the same in the $(s + 1)$ st iteration of the loop. Thus some x_i must be moved to another cluster. This cannot increase the total distance, because it decrease the distance to the current centers, and by the Lemma when we define the new centers in line 7 this can only decrease the total distance further. If the total distance stays the same, then x_i is moved to a cluster with smaller index, and the index sum decreases. \square

If the algorithm does not stop, there must be $s < t$ such that $C_s^j = C_t^j$ for all j . Then $\iota_t = \iota_s$. Moreover, $z_s^j = z_t^j$ for all j , which implies $\Delta_t = \Delta_s$.

By the claim, either $\Delta_t < \Delta_s$ or $\Delta_t = \Delta_s$ and $\iota_t < \iota_s$. This is a contradiction.



- ▶ The number of steps the κ -MEANS algorithm takes until it halts may be exponential in the number n of input points.
However, in practice the algorithm usually converges quickly.
- ▶ The κ -MEANS algorithm does not necessarily compute an optimal solution for the Centroid Clustering problem.
- ▶ The clustering the κ -MEANS algorithm returns may depend on the choice of the initial “centroids” $\mathbf{z}^1, \dots, \mathbf{z}^k$.

Complexity of Centroid Clustering

Centroid Clustering

Instance Points $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^\ell$, number $k \in \mathbb{N}$

Problem Find $\mathbf{z}^1, \dots, \mathbf{z}^k \in \mathbb{R}^\ell$ and a partition C^1, \dots, C^k of $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ that minimise $\sum_{j=1}^k \sum_{x \in C^j} \|\mathbf{x} - \mathbf{z}^j\|^2$

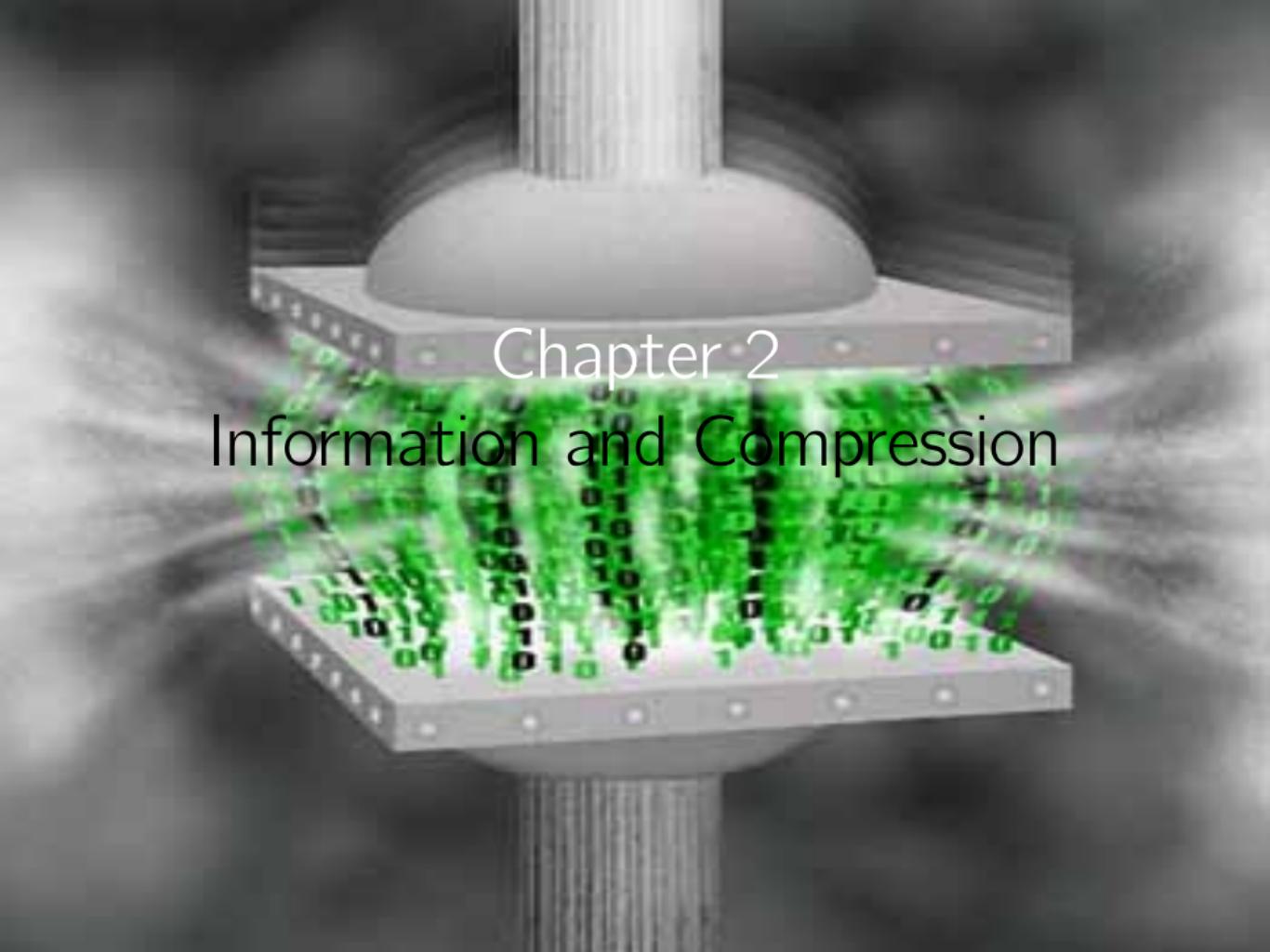
Theorem 1.19

- (1) *The Centroid Clustering problem is NP-hard, even if either the dimension ℓ or the cluster number k are fixed to be 2.*
- (2) *If both k and ℓ are fixed, the problem can be solved in polynomial time.*

(without proof)

References

[Russell and Norvig, 2021, Chapter 18] is a good (and reasonably short) introduction to general machine learning techniques. A more comprehensive introduction can be found in the textbook [Shalev-Shwartz and Ben-David, 2014]. Details on linear separators and clustering can be found in [Blum et al., 2020, Chapters 6 and 8] and [Shalev-Shwartz and Ben-David, 2014, Chapters 9, 15 and 22].



Chapter 2

Information and Compression

Goals of this Chapter

- ▶ Review basics of probability theory
- ▶ Discuss concentration inequalities
- ▶ Introduce the fundamentals of information theory, in particular entropy
- ▶ Discuss the relation between information and compression

2.1 Background from Probability Theory

Probability Spaces and Random Variables

Probability spaces

- ▶ Recall that a probability space consists of a set Ω , the **sample space**, a set $\mathcal{E} \subseteq 2^\Omega$, the **event space**, and a **probability distribution** \mathcal{P} that associates a probability $\mathcal{P}(E)$ with each event $E \in \mathcal{E}$.
- ▶ To avoid technicalities, in this course we usually assume our sample spaces Ω to be finite and the event spaces \mathcal{E} to be 2^Ω .

Notation

For an event $A(\omega)$ depending on an element $\omega \in \Omega$ of the sample space we sometimes write $\Pr_{\omega \sim \mathcal{P}}(A(\omega))$ to denote the probability $\mathcal{P}(A(\omega))$ of the event.

Review the exact definitions from your probability theory class!

Notes for Page 2.4

Recall that to specify a probability distribution \mathcal{P} on a finite (or countably infinite) sample space Ω , it suffices to specify a function

$$p : \Omega \rightarrow [0, 1]$$

with sum $\sum_{\omega \in \Omega} p(\omega) = 1$ and then let

$$\mathcal{P}(A) := \sum_{\omega \in A} p(\omega)$$

for all $A \subseteq \Omega$.

This does not work for continuous probability distributions, for example on $\Omega = \mathbb{R}$, where we usually have $\Pr(\{\omega\}) = 0$ for all $\omega \in \Omega$.

Random Variables

A **random variable** on a probability space (Ω, \mathcal{P}) is a mapping

$$X : \Omega \rightarrow \mathbb{R}.$$

- ▶ By $X \leq x$, for $x \in \mathbb{R}$, we denote the event $\{\omega \in \Omega \mid X(\omega) \leq x\}$. We usually denote the probability of this event by

$$\Pr(X \leq x),$$

without reference to the probability distribution \mathcal{P} .

- ▶ The function $F_X : \mathbb{R} \rightarrow [0, 1]$ defined by

$$F_X(x) := \Pr(X \leq x)$$

is known as the **cumulative distribution function** of X .

- ▶ We also use similar notations such as $X = x$ for the event $\{\omega \in \Omega \mid X(\omega) = x\}$ or $X \in A$ (for an $A \subseteq \mathbb{R}$) for the event $\{\omega \in \Omega \mid X(\omega) \in A\}$.

Example

Suppose our sample space Ω consists of the outcomes of rolling a dice. This may include the exact positions of the dice on or even under the table as well as which side is up. The probability distribution \mathcal{P} describing these outcomes may be fairly complicated.

A possible event $A = A(\omega)$ in this probability space is “the dice is on the table”.

Usually, we are not interested in the exact position of the dice on or under the table, but only in the number that comes up. Thus we may define a random variable $X : \Omega \rightarrow \{1, \dots, 6\}$ giving us this number. Then we would expect $\Pr(X = i) \approx \frac{1}{6}$ for all $i \in \{1, \dots, 6\}$.

Notes for Page 2.5

For random variables X with a finite or countably infinite range, we can define the **probability mass function** $f_X : \mathbb{R} \rightarrow [0, 1]$ by $f_X(x) := \Pr(X = x)$. Then we have

$$F_X(x) = \sum_{y \leq x} f_X(y).$$

(The sum is well defined, because $f_X(y) \neq 0$ for at most countably many y .) This gives us a probability distribution \mathcal{P}_X on \mathbb{R} defined by

$$\mathcal{P}_X(A) := \sum_{x \in A} \Pr(X = x).$$

(Again, the sum is well defined.) We call \mathcal{P}_X the **probability distribution** of X .

For **continuous** random variables (on an arbitrary probability space) there exists a **probability density function** $f_X : \mathbb{R} \rightarrow [0, 1]$ such that

$$F_X(x) = \int_{-\infty}^x f_X(y) dy.$$

In the continuous case, we can use the density function or the cumulative distribution function to define a probability distribution over the reals.

Random Vectors and Joint Distributions

Let (X, Y) be a pair of random variables with a finite range over the same probability space (Ω, \mathcal{P}) .

- ▶ The **joint distribution** of X, Y is the probability distribution $\mathcal{P}_{(X, Y)}$ on \mathbb{R}^2 defined by

$$\begin{aligned}\mathcal{P}_{(X, Y)}(\{(x, y)\}) &:= \Pr(X = x, Y = y) \\ &= \mathcal{P}(\{\omega \in \Omega \mid X(\omega) = x \text{ and } Y(\omega) = y\}).\end{aligned}$$

- ▶ Conversely, from the joint distribution we can retrieve the **marginal distribution** of X (and similarly of Y):

$$\Pr(X = x) = \sum_y \Pr((X, Y) = (x, y))$$

The definitions can easily be generalised to tuples (X_1, \dots, X_k) of random variables over the same probability space.

A tuple $\mathbf{X} = (X_1, \dots, X_k)$ of random variables is sometimes called a **random vector**.

Notes for Page 2.6

The definitions can be extended to infinite probability spaces.

A random vector is a mapping from a probability space to \mathbb{R}^k . More generally, a mapping X from a probability space (Ω, \mathcal{P}) to an arbitrary set \mathbb{U} is sometimes called a **random element**. The distribution of a random element is defined by

$$\Pr(X \in A) := \mathcal{P}(X^{-1}(A))$$

for a subset $A \subseteq \mathbb{U}$. (Note that for infinite spaces, $\mathcal{P}(X^{-1}(A))$ is not always defined, but only if it is in the event space. This issue needs to be taken care of, but it is beyond the scope of this course.)

Let (Ω, P) be a probability space and $k \geq 1$.

- ▶ Events $E_1, \dots, E_k \subseteq \Omega$ are **independent** if

$$\mathcal{P}(E_1, \dots, E_k) = \mathcal{P}(E_1) \cdots \mathcal{P}(E_k).$$

Here " E_1, \dots, E_k " denotes the event $E_1 \cap \dots \cap E_k$. Thus the Komma may be read as "and".

- ▶ Random variables $X_1, \dots, X_k : \Omega \rightarrow \mathbb{R}$ are **independent** if for all $A_1, \dots, A_k \subseteq \mathbb{R}$ the events $X_i \in A_i$ are independent, that is,

$$\Pr(X_1 \in A_1, \dots, X_k \in A_k) = \Pr(X_1 \in A_1) \cdots \Pr(X_k \in A_k).$$

We also say that the random vector $\mathbf{X} = (X_1, \dots, X_k)$ is independent.

Expectation and Variance

Let X be a random variable.

- ▶ The expectation of X is $E(X) = \sum_{x \in \mathbb{R}} x \cdot \Pr(X = x)$.
- ▶ The variance of X is $\text{Var}(X) = E((X - E(X))^2)$.

Properties

- (1) For $\alpha, \beta \in \mathbb{R}$, $E(\alpha X + \beta Y) = \alpha E(X) + \beta E(Y)$ (linearity of expectation).
- (2) If X and Y are independent then $E(X \cdot Y) = E(X)E(Y)$.
- (3) $\text{Var}(X) = E(X^2) - E(X)^2$.
- (4) If X and Y are independent then $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$.

Markov's and Chebyshev's Inequalities

Markov's Inequality

Let X be a nonnegative random variable. Then for all $a > 0$,

$$\Pr(X \geq a) \leq \frac{\mathbb{E}(X)}{a}.$$

Chebyshev's Inequality

Let X be a random variable. Then for all $b > 0$,

$$\Pr(|X - \mathbb{E}(X)| \geq b) \leq \frac{\text{Var}(X)}{b^2}.$$

Proof of Markov's Inequality.

$$\mathbb{E}(X) = \sum_{x \geq 0} x \cdot \Pr(X = x) \geq \sum_{x \geq a} x \cdot \Pr(X = x) \geq a \cdot \sum_{x \geq a} \Pr(X = x) = a \cdot \Pr(X \geq a).$$

and the inequality follows by dividing both sides by a . □

Proof of Chebyshev's Inequality.

Let $Y = (X - \mathbb{E}(X))^2$. Note that Y is a nonnegative random variable, and we have

$$\Pr(|X - \mathbb{E}(X)| \geq b) = \Pr(Y \geq b^2).$$

Moreover, $\mathbb{E}(Y) = \text{Var}(X)$. Thus by Markov's Inequality,

$$\Pr(|X - \mathbb{E}(X)| \geq b) = \Pr(Y \geq b^2) \leq \frac{\mathbb{E}(Y)}{b^2} = \frac{\text{Var}(X)}{b^2}. \quad \square$$

k -Wise Independence

A sequence X_1, \dots, X_n of random variables is **k -wise independent**, for some $k \leq n$, if for all $i_1 < \dots < i_k \in [n]$ the sequence X_{i_1}, \dots, X_{i_k} is independent.

Instead of 2-wise independent, we usually say **pairwise independent**.

Lemma 2.1

Let X_1, \dots, X_n be a pairwise independent sequence of random variables and $X := \sum_{i=1}^n X_i$. Then

$$\text{Var}(X) = \sum_{i=1}^n \text{Var}(X_i).$$

Proof of the Lemma.

Note first that for all $i \neq j$,

$$\begin{aligned}
 & E((X_i - E(X_i))(X_j - E(X_j))) \\
 &= E(X_i X_j) - 2 E(X_i) E(X_j) + E(X_i) E(X_j) \quad \text{by linearity of expectation} \quad (\star) \\
 &= E(X_i) E(X_j) - E(X_i) E(X_j) = 0 \quad \text{by independence.}
 \end{aligned}$$

In other words, the independent random variables X_i and X_j are uncorrelated.

Now a straightforward calculation proves the assertion:

$$\begin{aligned}
 & \text{Var}(X) \\
 &= E \left(\left(\sum_{i=1}^n (X_i - E(X_i)) \right)^2 \right) \quad \text{by linearity of expectation} \\
 &= E \left(\sum_{i \neq j} (X_i - E(X_i))(X_j - E(X_j)) + \sum_i (X_i - E(X_i))^2 \right) \\
 &= \sum_{i \neq j} E \left((X_i - E(X_i))(X_j - E(X_j)) \right) + \sum_i E \left((X_i - E(X_i))^2 \right) \quad \text{by linearity of expectation}
 \end{aligned}$$

$$= \sum_i \text{Var}(X_i) \quad \text{by } (\star).$$

□

The Weak Law of Large Numbers

Theorem 2.2

Let X_1, \dots, X_n be a pairwise independent sequence of random variables of variance $\text{Var}(X_i) \leq \sigma^2$, and let $X := \sum_{i=1}^n X_i$. Then for all $c > 0$,

$$\Pr(|X - E(X)| \geq cn) \leq \frac{\sigma^2}{c^2 n}.$$

Corollary 2.3 (Weak Law of Large Numbers)

Let X_1, X_2, \dots be a sequence of independent identically distributed random variables and $\mu := E(X_i)$. Then for all $\varepsilon > 0$,

$$\lim_{n \rightarrow \infty} \Pr\left(\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| \geq \varepsilon\right) = 0.$$

Proof of the Theorem.

$$\begin{aligned}\Pr(|X - \mathbb{E}(X)| \geq cn) &\leq \frac{\text{Var}(X)}{c^2 n^2} && \text{by Chebyshev's Inequality} \\ &= \frac{\sum_i \text{Var}(X_i)}{c^2 n^2} && \text{by Lemma 2.1} \\ &\leq \frac{n\sigma^2}{c^2 n^2} && \text{because } \text{Var}(X_i) \leq \sigma^2 \\ &= \frac{\sigma^2}{c^2 n}.\end{aligned}$$

2.2 Concentration Inequalities

Sums of Random Variables

- ▶ We often describe random processes by sums $X = \sum_{i=1}^n X_i$ of random variables.
- ▶ It is usually easy to compute the expected value $\mu := E(X)$.
- ▶ We want to show that with high probability, X is close to its expected value, that is,

$$\Pr(|X - \mu| \geq \text{something big}) \leq \text{something small.}$$

Such inequalities are called **concentration inequalities** or **tail bounds**.

- ▶ The weak law of large numbers (or rather Theorem 2.2) is a first, relatively weak concentration inequality.

In this section, we review (without proof) a few stronger concentration inequalities that we will need throughout this course.

Example 2.4

We roll a dice n times. We want to show that the number of '1's is close to $n/6$ for large n .

We let X_i be the indicator variable that a '1' comes up the i th time we roll the dice. That is, $X_i = 1$ if a '1' comes up and $X_i = 0$ otherwise.

We have $E(X_i) = \Pr(X_i = 1) = \frac{1}{6}$. Thus for $X := \sum_{i=1}^n X_i$ we have

$$E(X) = \sum_{i=1}^n E(X_i) = \frac{n}{6},$$

where the first equality holds by linearity of expectation.

It is plausible to assume that the X_i are independent random variables, which allows us to obtain very strong concentration inequalities fairly easily.

Examples (cont'd)

Example 2.5

We draw n balls from a bin with $N \geq n$ balls, of which K are black and the remaining $N - K$ are white. We want to estimate the number of black balls we draw.

We let X_i be the indicator random variable of the event that the i th ball we draw is black. Then $X = \sum_{i=1}^n X_i$ is the number of black balls we draw.

This process is more difficult to analyse, because the X_i are not identically distributed and not independent. In fact, the concentration inequalities we will see in this section will not apply to the setting of this example.

Chernoff Bounds

Chernoff Bounds (Multiplicative Version)

Let X_1, \dots, X_n be a sequence of independent $\{0, 1\}$ -valued random variables. Let $X := \sum_{i=1}^n X_i$ and $\mu := E(X)$. Then for $0 \leq c \leq 1$:

$$\Pr(X \geq (1 + c)\mu) \leq e^{-\frac{\mu c^2}{3}}$$

and

$$\Pr(X \leq (1 - c)\mu) \leq e^{-\frac{\mu c^2}{2}}.$$

Corollary 2.6

Let X_1, \dots, X_n be a sequence of independent $\{0, 1\}$ -valued random variables. Let $X := \sum_{i=1}^n X_i$ and $\mu := E(X)$. Then for $0 \leq c \leq 1$:

$$\Pr(|X - \mu| \geq c\mu) \leq 2e^{-\frac{\mu c^2}{3}}.$$

Hoeffding Bounds

Hoeffding Bounds

Let X_1, \dots, X_n be a sequence of independent identically distributed $\{0, 1\}$ -valued random variables. Let $X := \sum_{i=1}^n X_i$ and $\mu := E(X)$. Then for $0 \leq d \leq 1$:

$$\Pr(X \geq \mu + dn) \leq e^{-2nd^2}$$

and

$$\Pr(X \leq \mu - dn) \leq e^{-2nd^2}$$

Corollary 2.7

Let X_1, \dots, X_n be a sequence of independent identically distributed $\{0, 1\}$ -valued random variables. Let $X := \sum_{i=1}^n X_i$ and $\mu := E(X)$. Then for $0 \leq d \leq 1$:

$$\Pr(|X - \mu| \geq dn) \leq 2e^{-2nd^2}.$$

Concentration for More General Random Variables

Theorem 2.8

Let X_1, \dots, X_n be a sequence of independent random variables with $E(X_i) = 0$ and $\text{Var}(X_i) \leq \sigma^2$, and $X := \sum_{i=1}^n X_i$. Let $a \in \mathbb{R}$ such that $0 \leq a \leq \sqrt{2}n\sigma^2$ and suppose that

$$E(X_i^k) \leq \sigma^2 k!$$

for $3 \leq k \leq \left\lceil \frac{a^2}{4n\sigma^2} \right\rceil$. Then

$$\Pr(|X| \geq a) \leq 3e^{-\frac{a^2}{12n\sigma^2}}.$$

2.3 Entropy

Information of an Event

In a first step, we want to find a measure of the **information content** of a single event in a probability space in such a way that it only depends on the probability of the event.

The following requirements are natural:

- ▶ An event that is certain (has probability 1) has information content 0.
- ▶ An event that is impossible (has probability 0) has no information content. (That is, the information content is only defined for events of positive probability).
- ▶ Rarer events have higher information content.
- ▶ The joint information content of two independent events is the sum of their individual information contents. (That is, if A and B are independent events then the information content of $A \cap B$ is equal to the sum of the information content of A and the information content of B .)

Information of an Event (cont'd)

Lemma 2.9

Let $f : (0, 1] \rightarrow \mathbb{R}$ be a function satisfying the following conditions:

- (i) $f(1) = 0$;
- (ii) if $0 < x < y \leq 1$ then $f(x) > f(y)$;
- (iii) $f(xy) = f(x) + f(y)$ for all $x, y \in (0, 1]$.

Then there is a $b > 1$ such that $f(x) = \log_b \frac{1}{x}$ for all $x \in (0, 1]$.

Corollary 2.10

The only way to assign an information content $I(A)$ to events A of a finite probability space (Ω, \mathcal{P}) such that all our requirements are satisfied is by letting $I(A) = \log_b (1/\mathcal{P}(A))$ for some basis $b > 1$.

We decide to choose 2 as the basis of the logarithm, so that we can measure the information content in bits, and thus define the **information content** of an event A to be

$$\log \left(\frac{1}{\mathcal{P}(A)} \right).$$

Proof of the Lemma.

Let

$$c := f\left(\frac{1}{2}\right)$$

By (i) and (ii) we have $c > 0$. Thus by (iii), for all $p, q \in \mathbb{N}$ we have

$$f\left(\frac{1}{2^{\frac{p}{q}}}\right) = \frac{p}{q} \cdot c, \quad (*)$$

because $f(1/2^p) = p \cdot c$ and $q \cdot f(1/2^{p/q}) = f(1/2^p)$.

Claim

f is continuous.

Proof.

We need to prove that for every $x \in (0, 1]$ and every $\varepsilon > 0$ there is a $\delta > 0$ such that for all $y \in (0, 1]$, $|x - y| < \delta$ implies $|f(x) - f(y)| < \varepsilon$.

Let $x \in (0, 1]$ and $\varepsilon > 0$. Without loss of generality, let us assume that $x < 1$, which implies $f(x) > 0$ by (i), (ii). (The case $x = 1$ only requires a one-sided argument similar to ours.) Let us further assume that $f(x) - \varepsilon > 0$.

Choose $q \in \mathbb{N}$ such that $2c/q < \varepsilon$, and let $p \in \mathbb{N}$ such that $p \leq f(x) \cdot q/c < p + 1$. Then

$$f(x) - \varepsilon < \frac{p-1}{q}c < f(x) < \frac{p+1}{q}c < f(x) + \varepsilon.$$

By (\star) , we have

$$f\left(\frac{1}{2^{\frac{p-1}{q}}}\right) = \frac{p-1}{q} \cdot c \quad \text{and} \quad f\left(\frac{1}{2^{\frac{p+1}{q}}}\right) = \frac{p+1}{q} \cdot c.$$

It follows from (ii) that $x \in (1/2^{(p+1)/q}, 1/2^{(p-1)/q})$ and that for $y \in [1/2^{(p+1)/q}, 1/2^{(p-1)/q}]$ we have

$$f(x) - \varepsilon < \frac{p-1}{q}c \leq f(y) \leq \frac{p+1}{q}c < f(x) + \varepsilon$$

Letting $\delta := \min\{x - 1/2^{(p+1)/q}, 1/2^{(p-1)/q} - x\}$, we thus have

$$f(x) - \varepsilon < f(y) < f(x) + \varepsilon$$

for all $y \in (x - \delta, x + \delta)$.

□

Now choose $k \in \mathbb{N}$ such that $k \cdot c \geq 1$. Then $f(1/2^k) \geq 1$, and as $f(1) = 0$ and f is continuous, there is a $b \in (1, 2^k)$ such that $f(1/b) = 1$.

Then it follows from (i) and (iii) that for all $p \in \mathbb{N}, q \in \mathbb{Z}$ we have

$$f\left(\frac{1}{b^{p/q}}\right) = \frac{p}{q} = \log_b\left(b^{p/q}\right).$$

As f is continuous and the numbers $1/b^{p/q}$ for $p \in \mathbb{N}, q \in \mathbb{Z}$ are dense in $(0, 1]$, this implies $f(x) = \log_b(1/x)$ for all $x \in (0, 1]$. □

Definition 2.11

- (1) The **entropy** of a probability distribution \mathcal{P} on a finite sample space Ω is defined to be

$$H(\mathcal{P}) := \sum_{\omega \in \Omega} \mathcal{P}(\{\omega\}) \cdot \log \frac{1}{\mathcal{P}(\{\omega\})},$$

where we take $0 \cdot \log(1/0)$ to be 0 (or only sum over all ω with $\mathcal{P}(\omega) > 0$).

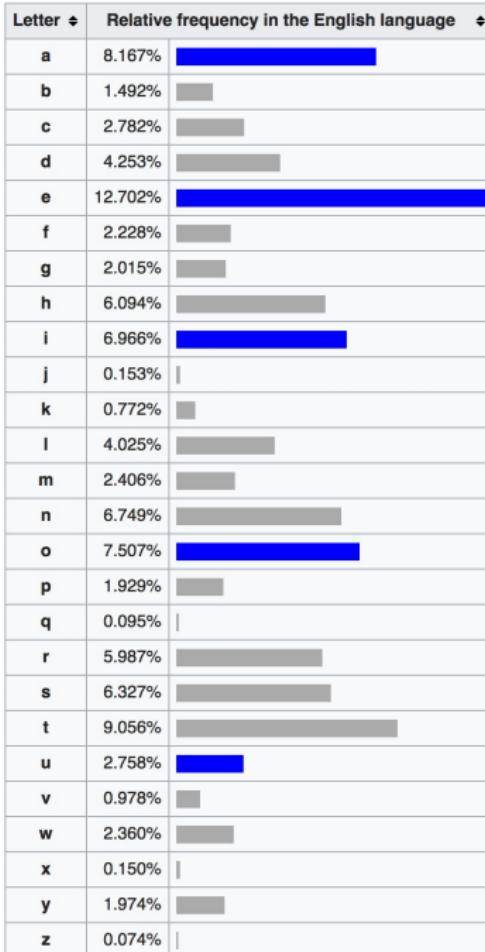
- (2) The **entropy** of a random variable X with finite range is defined to be

$$H(X) := \sum_{x \in \text{rg}(X)} \Pr(X = x) \cdot \log \frac{1}{\Pr(X = x)}.$$

Remark 2.12

The entropy is the expected value of the information content of the elementary events ω . It thus measures the average information we obtain by observing outcomes of the distribution \mathcal{P} .

Example



The table shows the letter frequencies in the English language (according to Wikipedia).

Let \mathcal{P} be the probability distribution on $\{a, \dots, z\}$ defined by the relative frequencies.

Then $H(\mathcal{P}) \approx 4.176$.

By comparison, the entropy of the uniform distribution \mathcal{U} on $\{a, \dots, z\}$ is

$$\begin{aligned} H(\mathcal{U}) &= \sum_{u \in \{a, \dots, z\}} \mathcal{U}(\{u\}) \log \left(\frac{1}{\mathcal{U}(\{u\})} \right) \\ &= \sum_{u \in \{a, \dots, z\}} \frac{1}{26} \log(26) \\ &= \log(26) \approx 4.700 \end{aligned}$$

Lower and Upper Bounds

Lemma 2.13

Let \mathcal{P} be a probability distribution on a sample space Ω of size $|\Omega| = n$. Then

$$0 \leq H(\mathcal{P}) \leq \log n.$$

Furthermore,

$$H(\mathcal{P}) = 0 \iff \exists \omega \in \Omega : \mathcal{P}(\{\omega\}) = 1$$

and

$H(\mathcal{P}) = \log(n) \iff \mathcal{P}$ is the uniform distribution, that is, $\mathcal{P}(\{\omega\}) = 1/n$ for all $\omega \in \Omega$.

Jensen's Inequality

Let $f : D \rightarrow \mathbb{R}$, where $D \subseteq \mathbb{R}$. Then f is **convex** if for all $\lambda \in [0, 1]$ and all $x, y \in D$ it holds that $\lambda x + (1 - \lambda)y \in D$ and

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

If $f(\lambda x + (1 - \lambda)y) < \lambda f(x) + (1 - \lambda)f(y)$ for all $\lambda \in (0, 1)$ and all distinct $x, y \in D$ then f is **strictly convex**.

Lemma 2.14 (Jensen's Inequality)

Let X be a random variable, and let $f : D \rightarrow \mathbb{R}$ be a convex function with $\text{rg}(X) \subseteq D$. Then $E(X) \in D$ and

$$f(E(X)) \leq E(f(X)).$$

If f is strictly convex, then equality holds if and only if X is constant.

Remark.

The condition $\lambda x + (1 - \lambda)y \in D$ in the definition of convex functions simply says that the domain D of f is a convex set, which for a set of reals means that it is an interval, possibly infinite on one or both sides.

Proof of Jensen's Inequality (for finite-valued X).

We prove the inequality by induction on $n := |\text{rg}(X)|$.

Suppose that $\text{rg}(X) = \{x_1, \dots, x_n\}$, and let $p_i := \Pr(X = x_i)$. Then $\sum_i p_i = 1$. For $n = 1$, we have $E(X) = x_1$ and $E(f(X)) = f(x_1)$. This is the case that X is constant, and indeed even equality holds.

For the inductive step $n \mapsto n + 1$, let $y := \frac{p_n}{p_n + p_{n+1}}x_n + \frac{p_{n+1}}{p_n + p_{n+1}}x_{n+1}$. Note that $y \in D$ and

$$f(y) \leq \frac{p_n}{p_n + p_{n+1}}f(x_n) + \frac{p_{n+1}}{p_n + p_{n+1}}f(x_{n+1})$$

by the convexity of f applied with $\lambda = \frac{p_n}{p_n + p_{n+1}}$. Moreover, if f is strictly convex then the inequality is strict.

We define a new random variable Y with $\text{rg}(Y) = \{x_1, \dots, x_{n-1}, y\}$ by $\Pr(Y = x_i) := p_i$ for $1 \leq i \leq n - 1$ and $\Pr(Y = y) := p_n + p_{n+1}$. Then by the induction hypothesis,

$$f(E(Y)) \leq E(f(Y)).$$

We have

$$E(Y) = \sum_{i=1}^{n-1} p_i x_i + (p_n + p_{n+1}) \cdot y = \sum_{i=1}^{n+1} p_i x_i = E(X)$$

and thus $f(E(X)) = f(E(Y))$. Moreover,

$$\begin{aligned} E(f(Y)) &= \sum_{i=1}^{n-1} p_i f(x_i) + (p_n + p_{n+1}) f(y) \\ &\leq \sum_{i=1}^{n-1} p_i f(x_i) + (p_n + p_{n+1}) \left(\frac{p_n}{p_n + p_{n+1}} f(x_n) + \frac{p_{n+1}}{p_n + p_{n+1}} f(x_{n+1}) \right) \\ &= \sum_{i=1}^{n+1} p_i f(x_i) = E(f(X)). \end{aligned}$$

If f is strictly convex, then the inequality is strict.

Thus $f(E(X)) = f(E(Y)) \leq E(f(Y)) \leq E(f(X))$, and the inequality is strict if f is strictly convex.



Log Sum Inequality

As a corollary to Jensen's Inequality, we get the following useful inequality.

Corollary 2.15 (Log Sum Inequality)

For all $i \in [n]$, let $p_i \in$, $q_i \in$, and let $p := \sum_i p_i$ and $q := \sum_i q_i$. Then

$$\sum_{i=1}^n p_i \log \left(\frac{p_i}{q_i} \right) \geq p \log \left(\frac{p}{q} \right)$$

Moreover, equality holds if and only if $p_i/q_i = p_j/q_j$ for all i, j .

Proof of the Log Sum Inequality.

Let

- ▶ (Ω, \mathcal{P}) be the probability space with $\Omega := [n]$ and \mathcal{P} defined by $P(\{i\}) := \frac{q_i}{q}$,
- ▶ $X : \Omega \rightarrow \mathbb{R}$ the random variable defined by $X(i) := \frac{p_i}{q_i}$,
- ▶ $f : \mathbb{R} \rightarrow \mathbb{R}$ the strictly convex function defined by $f(x) := x \log x$.

Then

$$\begin{aligned}\frac{1}{q} \sum_{i=1}^n p_i \log \left(\frac{p_i}{q_i} \right) &= \sum_{i=1}^n \frac{q_i}{q} \cdot \frac{p_i}{q_i} \cdot \log \left(\frac{p_i}{q_i} \right) \\ &= \sum_{i=1}^n P(\{i\}) \cdot f(X(i)) \\ &\geq f \left(\sum_{i=1}^n P(\{i\}) \cdot X(i) \right) \quad \text{by Jensen's Inequality}\end{aligned}$$

$$\begin{aligned}
&= f \left(\sum_{i=1}^n \frac{q_i}{q} \frac{p_i}{q_i} \right) \\
&= f \left(\frac{p}{q} \right) \\
&= \frac{1}{q} \cdot p \cdot \log \left(\frac{p}{q} \right).
\end{aligned}$$

Moreover, equality holds if and only if X is constant, or equivalently,
 $p_i/q_i = p_j/q_j$ for all i, j .



Proof of Lemma 2.13.

Suppose that $\Omega = \{\omega_1, \dots, \omega_n\}$, and let $p_i := \mathcal{P}(\{\omega_i\})$. Without loss of generality we may assume that $p_1 \geq p_2 \geq \dots \geq p_n$. Let $m \leq n$ such that $p_m > 0$ and $p_{m+1} = \dots = p_n = 0$. Then

$$H(\mathcal{P}) = \sum_{i=1}^m p_i \log \left(\frac{1}{p_i} \right)$$

If $m = 1$ then $p_1 = 1$ and thus $H(\mathcal{P}) = 1 \log 1 = 0$.

In the following, we assume that $m > 1$. Then for all $i \in [m]$ we have

$0 < p_i < 1$ and $\frac{1}{p_i} > 1$, which implies $\log \left(\frac{1}{p_i} \right) > 0$. This implies $H(\mathcal{P}) > 0$.

To prove the upper bound, note that by the Log Sum Inequality (with $q_i = \frac{1}{n}$) we have

$$\sum_{i=1}^n p_i \log(p_i/n) \geq 0,$$

with equality if and only if $p_i = \frac{1}{n}$ for all $i \in [n]$. Thus

$$H(\mathcal{P}) = \sum_{i=1}^m p_i \log \left(\frac{1}{p_i} \right)$$

$$\begin{aligned}
&= \sum_{i=1}^m p_i \log \left(\frac{n}{p_i n} \right) \\
&= \sum_{i=1}^m p_i \log n - \sum_{i=1}^n p_i \log(p_i n) \\
&\leq \log n \sum_{i=1}^m p_i \\
&= \log n,
\end{aligned}$$

with equality if and only if $p_i = \frac{1}{n}$ for all $i \in [n]$.



Interpretations of Entropy

Entropy as Information Content

So far, we have interpreted entropy as a measure for the average information content of a probability distribution or random variable. We will elaborate on this interpretation in the next section.

Entropy as Disarray

We may also view entropy as a measure of disarray:

- ▶ low entropy means that after drawing many samples from our distribution, we will not have much variation, in the extreme case of zero entropy we will always see the same element of the sample space;
- ▶ high entropy means that after drawing many samples we see many elements in arbitrary order, that is, complete chaos.

This is the physicists' interpretation of entropy.

Decision Tree Learning Revisited

Suppose we want to build a decision tree for a classification problem (see Section 1.3), and let \mathbb{Y} be the set of classes (target values).

- ▶ At each node of the tree we are given a set S of labelled examples (feature vectors labelled with a target value $y \in \mathbb{Y}$), and we must decide which of the features from a set \mathcal{A} we choose.
- ▶ If we choose feature A with domain \mathbb{D}_A , then for each $x \in \mathbb{D}_A$ we create a child of the current node and pass the set $S_{A,x}$ of all examples in S whose A -value is x to this child.
- ▶ It is our strategy to choose the feature that “discriminates best” between the examples in S .
Intuitively, this means that we want to reduce the disarray of target values at the children of the current node.

Using entropy, we can now formalise this intuition.

Information Gain

Let S be a set of labelled examples (\mathbf{x}, y) , where \mathbf{x} is a feature vector over \mathcal{A} and $y \in \mathbb{Y}$ a target value.

- We let \mathcal{P} be the probability distribution on \mathbb{Y} defined by

$$\mathcal{P}(\{y\}) := \frac{|\{(\mathbf{x}, y') \in S \mid y' = y\}|}{|S|}.$$

That is, $\mathcal{P}(\{y\})$ is the probability that an example chosen uniformly at random from S has target value y .

- For each attribute A and value x in the domain \mathbb{D}_A of A we let $S_{A=x}$ be the set of all examples in S with A -value x .
We let $\mathcal{P}_{A=x}$ be the probability distribution on \mathbb{Y} defined by

$$\mathcal{P}_{A=x}(\{y\}) := \frac{|\{(\mathbf{x}, y') \in S_{A=x} \mid y' = y\}|}{|S_{A=x}|}.$$

That is, $\mathcal{P}_{A=x}(\{y\})$ is the probability that an example chosen uniformly at random from $S_{A=x}$ has target value y .

Information Gain (cont'd)

The **information gain** of feature A is the average difference between the entropies of \mathcal{P} and $\mathcal{P}_{A=x}$, weighted by the relative size of $S_{A=x}$:

$$G(S, A) := H(\mathcal{P}) - \sum_{x \in \mathbb{D}_A} \frac{|S_{A=x}|}{|S|} H(\mathcal{P}_{A=x}).$$

Intuitively, it measures the reduction in disarray we can achieve by choosing attribute A . In building the decision tree, it is a good strategy to choose the attribute A maximising $G(S, A)$.

Notes for Page 2.30

Example

Example No	W	D	P	T	S	L	Output
1	Sunny	Fri	no	ML	11-15	Valiant	yes
2	Cloudy	Fri	yes	ML	16-20	Valiant	yes
3	Cloudy	Fri	yes	ML	11-15	Valiant	no
4	Sunny	Fri	no	ML	11-15	Valiant	yes
5	Cloudy	Fri	no	ML	16-20	Valiant	yes
6	Snow	Fri	no	ML	16-20	Valiant	no
7	Rainy	Fri	no	ML	16-20	Valiant	yes
8	Rainy	Fri	no	ML	16-20	Valiant	yes
9	Rainy	Fri	no	ML	8-10	Valiant	no
10	Sunny	Fri	no	ML	8-10	Valiant	no

We are in the setting of the “attending class” example of Section 1.3. We use our decision tree algorithm and are at a node of the tree where we have

already set the attributes **D**ay to “Friday”, **T**opic to “ML”, and **L**ecturer to “Valiant”.

The set of remaining attributes is

$$\mathcal{A} := \{\textbf{W}eather, \textbf{P}arty the night before, \textbf{S}tart time\}.$$

The table above shows our remaining data set S .

- ▶ The random variable Y takes value “yes” with probability 6/10 and value “no” with probability 4/10.

Thus

$$H(Y) = \frac{6}{10} \log \frac{10}{6} + \frac{4}{10} \log \frac{10}{4} \approx 0.971.$$

For the attribute **W**, we get the following:

- ▶ The random variable $Y_{W,\text{Sunny}}$ takes value “yes” with probability 2/3 and value “no” with probability 1/3.
- ▶ The random variable $Y_{W,\text{Cloudy}}$ takes value “yes” with probability 2/3 and value “no” with probability 1/3.

- ▶ The random variable $Y_{W,\text{Rainy}}$ takes value “yes” with probability 2/3 and value “no” with probability 1/3.
- ▶ The random variable $Y_{W,\text{Snow}}$ takes value “no” with probability 1.

Thus we have

$$\begin{aligned} H(Y_{W,\text{Sunny}}) &\approx 0.918 \\ H(Y_{W,\text{Cloudy}}) &\approx 0.918 \\ H(Y_{W,\text{Rainy}}) &\approx 0.918 \\ H(Y_{W,\text{Snow}}) &= 0. \end{aligned}$$

The information gain of **W** is

$$\begin{aligned} G(S, \mathbf{W}) = H(Y) - & \left(\frac{3}{10} H(Y_{W,\text{Sunny}}) \right. \\ & + \frac{3}{10} H(Y_{W,\text{Cloudy}}) \\ & + \frac{3}{10} H(Y_{W,\text{rainy}}) \\ & \left. + \frac{1}{10} H(Y_{W,\text{Snow}}) \right) \approx 0.144 \end{aligned}$$

For the attribute **P**, we get the following:

- ▶ The random variable $Y_{P,\text{yes}}$ takes value “yes” with probability 1/2 and value “no” with probability 1/2.
- ▶ The random variable $Y_{P,\text{no}}$ takes value “yes” with probability 5/8 and value “no” with probability 3/8.

Thus we have

$$H(Y_{P,\text{Yes}}) \approx 0.954$$

$$H(Y_{P,\text{No}}) = 1$$

and

$$G(S, \mathbf{P}) \approx 0.007.$$

Finally, for the attribute **S**, we get the following:

- ▶ The random variable $Y_{S,8-10}$ takes value “no” with probability 1.
- ▶ The random variable $Y_{S,11-15}$ takes value “yes” with probability 2/3 and value “no” with probability 1/3.

- The random variable $Y_{S,16-20}$ takes value “yes” with probability 4/5 and value “no” with probability 1/5.

Thus we have

$$H(Y_{S,8-10}) = 0$$

$$H(Y_{S,11-15}) \approx 0.918$$

$$H(Y_{S,16-20}) \approx 0.722$$

and

$$G(S, \mathbf{S}) \approx 0.334.$$

Thus **S** is the attribute with the highest information gain.

2.4 Compression

Information and Compression

What is the meaning of “Information”?

Our intuition behind the entropy of a probability distribution \mathcal{P} is that it measures the average “information” of a sample from \mathcal{P} , that is, of an elementary event $\{\omega\}$.

But what is this supposed to mean?

Here are two possible explanations:

- ▶ Information is the average number of bits we need to transfer (or store) many consecutive samples from a distribution, using the best possible coding scheme for this specific distribution.
- ▶ Suppose we have a string where the symbols are independently sampled from our distribution. Then the information content of the distribution should measure how well we can compress the string.

In both scenarios, the essence is to explain **information as a measure for compressibility**. In this section, we will prove **Shannon's Source Coding Theorem**, which states that indeed entropy is a measure for compressibility.

We want to compress strings over a finite **source alphabet** Σ . We always assume that $|\Sigma| \geq 2$. We encode the compressed strings in binary, thus our **target alphabet** is $\{0, 1\}$.

A **compression scheme** over Σ is a pair $\Gamma = (\text{com}_\Gamma, \text{dec}_\Gamma)$ consisting of a **compression mapping** $\text{com}_\Gamma : \Sigma^* \rightarrow \{0, 1\}^*$ and a **decompression mapping** $\text{dec}_\Gamma : \{0, 1\}^* \rightarrow \Sigma^*$.

Notation

If Γ is clear from the context we often omit the index Γ and just write **com** and **dec**.

Goal

Ideally, we would like to have a compression scheme that is **lossless**, that is, $\text{dec}(\text{com}(\mathbf{x})) = \mathbf{x}$ for all $\mathbf{x} \in \Sigma^*$, and guarantees a good **compression rate**.

Compression Rate

The compression rate of a scheme Γ on a string \mathbf{x} is $\frac{|\text{com}(\mathbf{x})|}{|\mathbf{x}|}$.

Intuitively, the compression rate of Γ is the maximum of the compression rates of Γ on all strings in Σ^* . As this maximum does not necessarily exist, we define a compression rate for strings of length n separately for each n :

the compression rate of Γ is the function $\rho_\Gamma : \mathbb{N} \rightarrow \mathbb{R}$ defined by

$$\rho_\Gamma(n) := \max_{\mathbf{x} \in \Sigma^n} \frac{|\text{com}(\mathbf{x})|}{|\mathbf{x}|}.$$

Remark

The compression rate is somewhat unfair as a measure of compression if we do not take the size of the source alphabet Σ into account. Σ may be much larger than the target alphabet $\{0, 1\}$, and thus source strings can be much shorter.

However, ultimately the symbols of Σ also have to be encoded as bit strings, and (without any compression) this requires $\lceil \log |\Sigma| \rceil$ bits per symbol.

Thus we may argue that a compression scheme Γ achieves an actual compression if $com(\mathbf{x}) < |\mathbf{x}| \cdot \lceil \log |\Sigma| \rceil$ for all \mathbf{x} , or equivalently $\rho_\Gamma(n) < \lceil \log |\Sigma| \rceil$.

Guaranteed Lossless Compression is Impossible

Observation 2.16

Let $n \in \mathbb{N}$. There is no lossless compression scheme Γ such that $\rho_\Gamma(n) < \log |\Sigma|$.

Consequence

We need to compromise either on the losslessness or on the compression rate.

- ▶ We may consider lossy compression schemes that guarantee a small compression rate, but risk information loss.
- ▶ We may consider lossless schemes that sometimes generate “compressed” strings that are longer than the original strings.

Our goal should be to make information loss/bad compression rate on a string unlikely. To be able to do this in a meaningful way and quantify our risk, we must make assumptions about the input strings.

Proof.

We can show the observation with a simple counting argument: Assume that there is a lossless compression scheme Γ with

$$\rho_{\Gamma}(n) = \max_{x \in \Sigma^n} \frac{|\text{com}(x)|}{|x|} < \log |\Sigma|.$$

Thus

$$\Gamma : \Sigma^n \rightarrow S := \{y \in \{0, 1\}^* \mid |y| < n \log |\Sigma|\}$$

is an injection. Then $|S|$ must be greater or equal to $|\Sigma^n|$. But

$$|S| < 2^{n \log |\Sigma|} = |\Sigma|^n = |\Sigma^n|,$$

leading to a contradiction.

Generating the Input Strings

We assume that the input strings are generated randomly.

We let \mathcal{P} be a probability distribution on Σ . We assume that symbol x_i in a string $\mathbf{x} = x_1 \dots x_n \in \Sigma^*$ is sampled from \mathcal{P} . We assume the individual samples to be independent.

This is known as the **i.i.d. assumption** (independent identically distributed).

Formally, for every $n \in \mathbb{N}$ we define a probability distribution \mathcal{P}^n on Σ^n by

$$\mathcal{P}^n(\{x_1 \dots x_n\}) := \prod_{i=1}^n \mathcal{P}(\{x_i\}).$$

Then in our model, strings of length n are distributed according to \mathcal{P}^n .

Note that this does **not** give us a probability distribution on the set Σ^* of all strings over Σ .

Example

\mathcal{P} may be the distribution over the ASCII (or UTF8) alphabet, where the probability of a symbol is proportional to its frequency in English texts, and our goal is to compress a text file.

Note: Usually the i.i.d. assumption does not hold in this scenario.

Proviso

Until the rest of this section, we always consider strings over a finite alphabet Σ that are generated by a probability distribution \mathcal{P} under the i.i.d. assumption.

We denote the resulting probability distribution on strings $\mathbf{x} \in \Sigma^n$ by \mathcal{P}^n .

Let Γ be a compression scheme over Σ .

We define the **loss rate** of Γ to be the probability that a compressed string is not decompressed correctly.

As we only have a probability distribution on strings of length n , we define the loss rate as a function of n :

$$\lambda_{\Gamma, \mathcal{P}}(n) := \Pr_{\mathbf{x} \sim \mathcal{P}^n} (\mathbf{x} \neq \text{dec}(\text{com}(\mathbf{x})))$$

Optimal Compression

Lossy Compression

We shall design a lossy compression scheme that reaches the optimal compression rate and has a negligible loss rate.

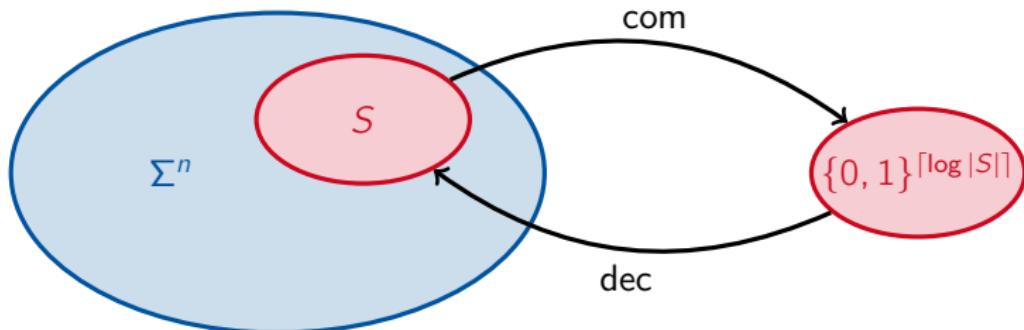
Lossless Compression

It is also possible to design a lossless compression scheme with an expected compression rate close to the optimum.

Shannon's Source Coding Theorem

In both cases, the optimal compression rate is the entropy of the probability distribution \mathcal{P} .

Lossy Compression (Idea)



Key Observation

For sufficiently large n , the most likely, or “typical”, strings in Σ^n fall into a relatively small set S .

Compression Scheme

We focus on the strings in the set S and compress them losslessly into binary strings of length $\{0, 1\}^{\lceil \log |S| \rceil}$.

We ignore the remaining strings in $\Sigma^n \setminus S$ and write them off as losses.

Notes for Page 2.39

Example

Suppose that $\Sigma = \{a, b, \dots, h\}$ and that \mathcal{P} is defined as follows:

x	a	b	c	d	e	f	g	h
$\mathcal{P}(\{x\})$	$3/8$	$5/16$	$3/32$	$1/16$	$1/16$	$1/32$	$1/32$	$1/32$

We have $H(\mathcal{P}) \approx 2.34$.

Let us first compress the strings of length 1. If we accept a loss rate of $5/32 \approx 0.16$, we can use the following compression scheme of compression rate 2 (compare this with $\log |\Sigma| = 3$).

Compression:

x	a	b	c	d	e	f	g	h
$\text{com}(x)$	00	01	10	11	00	00	00	00

Decompression:

y	00	01	10	11
$\text{dec}(y)$	a	b	c	d

Let us verify that the loss rate is 5/32:

$$\lambda_{\Gamma}(1) = \Pr_{x \in \mathcal{P}^1}(x \neq \text{dec}(\text{com}(x))) = \mathcal{P}^1(\{e, f, g, h\}) = \frac{5}{32}.$$

It should also be clear that we cannot get a compression rate below the trivial 3 with a loss rate below 5/32.

Let us now encode strings of length 2. Say, we want to achieve a loss rate of 1/10.

Let

$$S = \{ aa, ab, ba, bb, ac, ca, bc, cb, ad, ae, da, ea, bd, be, db, eb, af, ag, ah, fa, ga, ha, bf, bg, bh, fb, gb, hb \}$$

We have $|S| = 28$, and a straightforward calculation shows that $\mathcal{P}^2(S) \approx 0.9023$.

We define a 5-bit compression scheme by mapping S injectively into $\{0, 1\}^5$ and extending the mapping arbitrarily to the remaining strings in Σ^2 :

Compression:

x	aa	ab	ba	bb	\dots	gb	hb	cc	\dots
$\text{com}(x)$	00000	00001	00010	00011	\dots	11010	11011	11100	\dots

Decompression:

y	00000	00001	00010	00011	\dots	11010	11011	11100	\dots
$\text{dec}(y)$	aa	ab	ba	bb	\dots	gb	hb	cc	\dots

The compression rate of the scheme is $5/2$, and the loss rate is at most $1 - \mathcal{P}^2(S) < 1/10$.

We can continue like this on strings of increasing length and hope to get compression schemes with better and better compression rates and/or better loss rates. It turns out, however, that the entropy of \mathcal{P} is a limit for the compression rate, even if we keep the bound on the loss rate constant, say, at 0.1.

What we can do is improve the loss rate arbitrarily close to 0, as long as the compression rate remains above the entropy.

Compression

For every $\varepsilon > 0$ we define a compression scheme $\Gamma_\varepsilon = (\text{com}_\varepsilon, \text{dec}_\varepsilon)$.

- ▶ For every $n \in \mathbb{N}$, we choose a set $S_\varepsilon(n) \subseteq \Sigma^n$ of minimum cardinality such that

$$\mathcal{P}^n(S_\varepsilon(n)) \geq 1 - \varepsilon.$$

Let $s_\varepsilon(n) := \lceil \log |S_\varepsilon(n)| \rceil$.

- ▶ We define the compression mapping com_ε in such a way that for every n we have

$$\text{com}_\varepsilon(\Sigma^n) \subseteq \{0, 1\}^{s_\varepsilon(n)}$$

and that the restriction of com_ε to $S_\varepsilon(n)$ is injective.

- ▶ We define the decompression mapping dec_ε in such a way that for all $\mathbf{x} \in S_\varepsilon(n)$ we have $\text{dec}_\varepsilon(\text{com}_\varepsilon(\mathbf{x})) = \mathbf{x}$.

Loss Rate and Compression Rate

Let $\lambda_\varepsilon(n) := \lambda_{\Gamma_\varepsilon, \mathcal{P}}(n)$ be the loss rate of Γ_ε and $\rho_\varepsilon(n) := \rho_{\Gamma_\varepsilon}(n)$ the compression rate.

Loss rate

For all $n \in \mathbb{N}$,

$$\lambda_\varepsilon(n) \leq \varepsilon.$$

Compression rate

For all $n \in \mathbb{N}$,

$$\rho_\varepsilon(n) = \frac{s_\varepsilon(n)}{n}.$$

Both assertions are simple observations:

Loss rate

$$\lambda_\varepsilon(n) = \Pr_{x \sim \mathcal{P}^n} (\text{dec}_\varepsilon(\text{com}_\varepsilon(x)) \neq x) \leq \mathcal{P}^n(\Sigma^n \setminus S_\varepsilon(n)) \leq \varepsilon.$$

Compression rate

By the definition of com_ε , for all $x \in \Sigma^n$ we have $\text{com}_\varepsilon(x) \in \{0, 1\}^{s_\varepsilon(n)}$ and thus

$$\frac{|\text{com}_\varepsilon(x)|}{|x|} = \frac{s_\varepsilon(n)}{n}.$$

Main Lemma

Lemma 2.17

$$\lim_{n \rightarrow \infty} \frac{s_\varepsilon(n)}{n} = H(\mathcal{P}).$$

Proof.

We fix some notation first. Throughout the proof, we let

$$\begin{aligned}S &:= S_\varepsilon(n) \\s &:= s_\varepsilon(n) = \lceil \log |S| \rceil\end{aligned}$$

(for a value n depending on the context), and

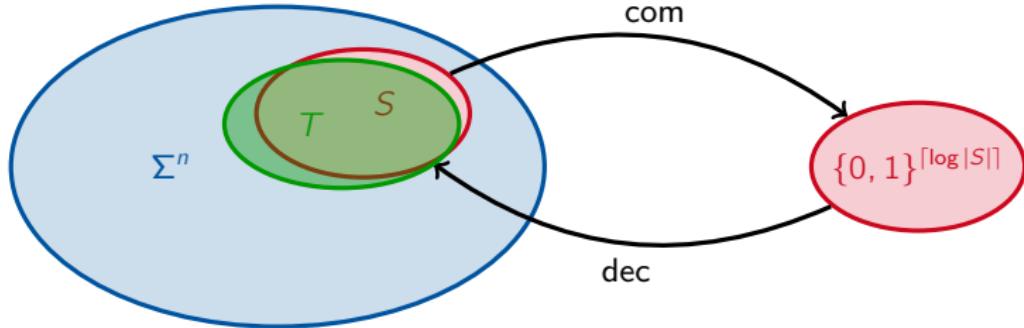
$$H := H(\mathcal{P}).$$

For every $\beta > 0$, we let

$$T_\beta(n) := \left\{ \mathbf{x} \in \Sigma^n \mid 2^{-n(H+\beta)} < \mathcal{P}^n(\{\mathbf{x}\}) < 2^{-n(H-\beta)} \right\}.$$

$T_\beta(n)$ is sometimes called the **typical set** (with parameter β). Arguably, it contains the \mathbf{x} in the "typical" probability range.

Intuitively, the set $T := T_\beta(n)$ is close to S :



We define a random variable Y_n on the probability space $(\Sigma^n, \mathcal{P}^n)$ by

$$Y_n(\mathbf{x}) := \log \left(\frac{1}{\mathcal{P}^n(\{\mathbf{x}\})} \right).$$

Claim 1

$$E(Y_n) = n \cdot H$$

Proof.

We have

$$E(Y_n) = \sum_{x \in \Sigma^n} \mathcal{P}^n(\{x\}) \log \left(\frac{1}{\mathcal{P}^n(\{x\})} \right) = H(\mathcal{P}^n).$$

This proves the claim for $n = 1$. For $n \geq 1$, we decompose $H(\mathcal{P}^n)$ using the independence assumption:

$$\begin{aligned} H(\mathcal{P}^n) &= \sum_{x \in \Sigma^n} \mathcal{P}^n(\{x\}) \log \left(\frac{1}{\mathcal{P}^n(\{x\})} \right) \\ &= \sum_{x' \in \Sigma^{n-1}} \sum_{x_n \in \Sigma} \mathcal{P}^{n-1}(\{x'\}) \mathcal{P}(\{x_n\}) \log \left(\frac{1}{\mathcal{P}^{n-1}(\{x'\}) \mathcal{P}(\{x_n\})} \right) \\ &= \sum_{x' \in \Sigma^{n-1}} \sum_{x_n \in \Sigma} \mathcal{P}^{n-1}(\{x'\}) \mathcal{P}(\{x_n\}) \log \left(\frac{1}{\mathcal{P}^{n-1}(\{x'\})} \right) \\ &\quad + \sum_{x' \in \Sigma^{n-1}} \sum_{x_n \in \Sigma} \mathcal{P}^{n-1}(\{x'\}) \mathcal{P}(\{x_n\}) \log \left(\frac{1}{\mathcal{P}(\{x_n\})} \right) \\ &= \sum_{x' \in \Sigma^{n-1}} \mathcal{P}^{n-1}(\{x'\}) \log \left(\frac{1}{\mathcal{P}^{n-1}(\{x'\})} \right) \underbrace{\sum_{x_n \in \Sigma} \mathcal{P}(\{x_n\})}_{=1} \end{aligned}$$

$$\begin{aligned}
& + \sum_{x_n \in \Sigma} \mathcal{P}(\{x_n\}) \log \left(\frac{1}{\mathcal{P}(\{x_n\})} \right) \underbrace{\sum_{x' \in \Sigma^{n-1}} \mathcal{P}^{n-1}(\{x'\})}_{=1} \\
& = H(\mathcal{P}^{n-1}) + H(\mathcal{P}).
\end{aligned}$$

Now the claim follows by induction. \square

Let $V := \text{Var}(Y_1)$.

Claim 2

$$\text{Var}(Y_n) = n \cdot V.$$

Proof.

For $\mathbf{x} = (x_1, \dots, x_n) \in \Sigma^n$, we have

$$Y_n(\mathbf{x}) = \sum_{i=1}^n \log \left(\frac{1}{\mathcal{P}(\{x_i\})} \right).$$

Thus Y_n is the sum of n identically distributed random variables X_i defined by $X_i(x) := \log(1/\mathcal{P}(\{x\}))$. Thus

$$\text{Var}(Y_n) = \text{Var}\left(\sum_{i=1}^n \text{Var}(X_i)\right) = \sum_{i=1}^n \text{Var}(X_i).$$

Note that $X_1 = Y_1$ and hence $\text{Var}(X_i) = \text{Var}(X_1) = V$. The claim follows. \square

Claim 3

$$\mathcal{P}^n(T_\beta(n)) = 1 - \frac{V}{n\beta^2}$$

Proof.

We have

$$\begin{aligned} \mathbf{x} \in T_\beta(n) &\iff 2^{-n(H+\beta)} < \mathcal{P}^n(\{\mathbf{x}\}) < 2^{-n(H-\beta)} \\ &\iff n(H-\beta) < \log \frac{1}{\mathcal{P}^n(\{\mathbf{x}\})} < n(H+\beta) \\ &\iff \left| \log \left(\frac{1}{\mathcal{P}^n(\{\mathbf{x}\})} \right) - nH \right| < n\beta \end{aligned}$$

$$\iff |Y_n - E(Y_n)| < n\beta.$$

By Chebyshev's Inequality, we have

$$\Pr(|Y_n - E(Y_n)| < n\beta) \geq 1 - \frac{\text{Var}(Y_n)}{n^2\beta^2} = \frac{V}{n\beta^2}$$

□

Claim 4

For sufficiently large n we have $\frac{s}{n} < H + 2\beta$.

Proof.

Suppose that $n \geq \frac{V}{\epsilon\beta^2}$. Then $\frac{V}{n\beta^2} \leq \epsilon$ and thus, by Claim 3,

$$\mathcal{P}^n(T_\beta(n)) \geq 1 - \epsilon.$$

As S is a set of minimum size with $\mathcal{P}^n(S) \geq 1 - \epsilon$, it follows that $|T_\beta(n)| \geq |S|$.

For all $\mathbf{x} \in T_\beta(n)$ we have $\mathcal{P}^n(\{\mathbf{x}\}) > 2^{-n(H+\beta)}$. Thus

$$|T_\beta(n)|2^{-n(H+\beta)} < \sum_{\mathbf{x} \in T_\beta(n)} \mathcal{P}^n(\{\mathbf{x}\}) \leq 1,$$

and this implies

$$|S| \leq |T_\beta(n)| \leq 2^{n(H+\beta)}$$

It follows that

$$s = \lceil \log |S| \rceil \leq \lceil n(H + \beta) \rceil.$$

For sufficiently large n , we have $\lceil n(H + \beta) \rceil \leq n(H + 2\beta)$. This implies the claim. \square

Claim 5

For sufficiently large n , we have $\frac{s}{n} > H - 2\beta$.

Proof.

To simplify the notation, let $T := T_\beta(n)$. Suppose for contradiction that $s \leq n(H - 2\beta)$. Then

$$|S| \leq 2^{n(H-2\beta)}.$$

To bound the probability of S we split it as follows:

$$\mathcal{P}^n(S) = \mathcal{P}^n(S \cap T) + \mathcal{P}^n(S \setminus T).$$

We have

$$\mathcal{P}^n(S \cap T) < |S|2^{-n(H-\beta)} \leq 2^{n(H-2\beta)-n(H-\beta)} = 2^{-n\beta},$$

where the first inequality holds because $\mathcal{P}^n(\{\mathbf{x}\}) < 2^{-n(H-\beta)}$ for all $\mathbf{x} \in T$. Moreover,

$$\mathcal{P}^n(S \setminus T) \leq 1 - \mathcal{P}^n(T) \leq \frac{V}{n\beta^2}.$$

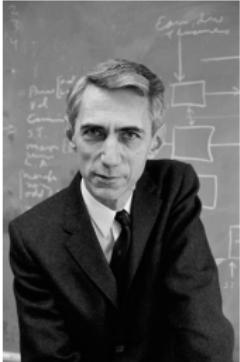
Thus

$$\mathcal{P}^n(S) < 2^{-n\beta} + \frac{V}{n\beta^2}$$

For sufficiently large n , this is smaller than $1 - \varepsilon$. However, we have $\mathcal{P}^n(S) \geq 1 - \varepsilon$ by the choice of S . This contradiction proves the claim. \square

Claims 4 and 5 imply the assertion of the lemma. \square

Shannon's Source Coding Theorem



Claude Shannon (1918-2001)

Photo: Alfred Eisenstaedt/ The LIFE Picture Collection/ Getty Images

Theorem 2.18 (Source Coding Theorem)

- (1) For every $\varepsilon > 0$ there is a compression scheme Γ_ε over Σ such that $\lambda_{\Gamma_\varepsilon, \mathcal{P}}(n) \leq \varepsilon$ for all n and $\lim_{n \rightarrow \infty} \rho_{\Gamma_\varepsilon}(n) = H(\mathcal{P})$.
- (2) There is no compression scheme Γ such that, for some $\alpha, \beta > 0$, it holds that $\lambda_{\Gamma, \mathcal{P}}(n) \leq 1 - \alpha$ and $\rho_{\Gamma}(n) \leq H(\mathcal{P}) - \beta$ for infinitely many $n \in \mathbb{N}$.

Proof.

We have already proved assertion (1) for the compression scheme Γ_ε introduced before.

Essentially, we have also proved (2), although this may not be obvious at first sight.

Let $\alpha, \beta > 0$. Suppose for contradiction that $\Gamma = (\text{com}, \text{dec})$ is a compression scheme over Σ with $\lambda_{\Gamma, \mathcal{P}}(n) \leq 1 - \alpha$ and $\rho_\Gamma(n) \leq H(\mathcal{P}) - \beta$ for infinitely many $n \in \mathbb{N}$.

Let $\varepsilon := 1 - \alpha$. By Lemma 2.17 there is an $n_0 \in \mathbb{N}$ such that

$$\frac{s_\varepsilon(n)}{n} > H(\mathcal{P}) - \beta/2$$

for all $n \geq n_0$.

Let $n_1 := \max\{n_0, \lceil 2/\beta \rceil\}$. Then $n \cdot \beta/2 \geq 1$ for all $n \geq n_1$.

Now we choose an $n \geq n_1$ such that $\lambda_{\Gamma, \mathcal{P}}(n) \leq 1 - \alpha$ and $\rho_{\Gamma_\varepsilon}(n) \leq H(\mathcal{P}) - \beta$. Let S be the set of all $\mathbf{x} \in \Sigma_n$ such that $\text{dec}(\text{com}(\mathbf{x})) = \mathbf{x}$. Then

$$\mathcal{P}^n(\Sigma^n \setminus S) = \lambda_{\Gamma, \mathcal{P}}(n) \leq \varepsilon$$

and therefore $\mathcal{P}^n(S) \geq 1 - \varepsilon$. By the definition of $S_\varepsilon(n)$ we have $|S_\varepsilon(n)| \leq |S|$ and thus

$$n(H(\mathcal{P}) - \beta/2) < s_\varepsilon(n) \leq \lceil \log |S| \rceil. \quad (*)$$

Let $r := n \cdot \rho_{\Gamma_\varepsilon}(n)$. Then $\text{com}(\mathbf{x}) \in \bigcup_{i=0}^r \{0, 1\}^i$ for all $\mathbf{x} \in \Sigma^n$. By the definition of S , the restriction of com to the set S must be injective, and thus

$$|S| \leq \left| \bigcup_{i=0}^r \{0, 1\}^i \right| = \sum_{i=0}^r 2^i < 2^{r+1}.$$

It follows that

$$\lceil \log |S| \rceil \leq r + 1 = n \cdot \rho_{\Gamma_\varepsilon}(n) + 1 \leq n \cdot (H(\mathcal{P}) - \beta) + 1.$$

As $n \cdot \beta/2 \geq 1$, this contradicts $(*)$. □

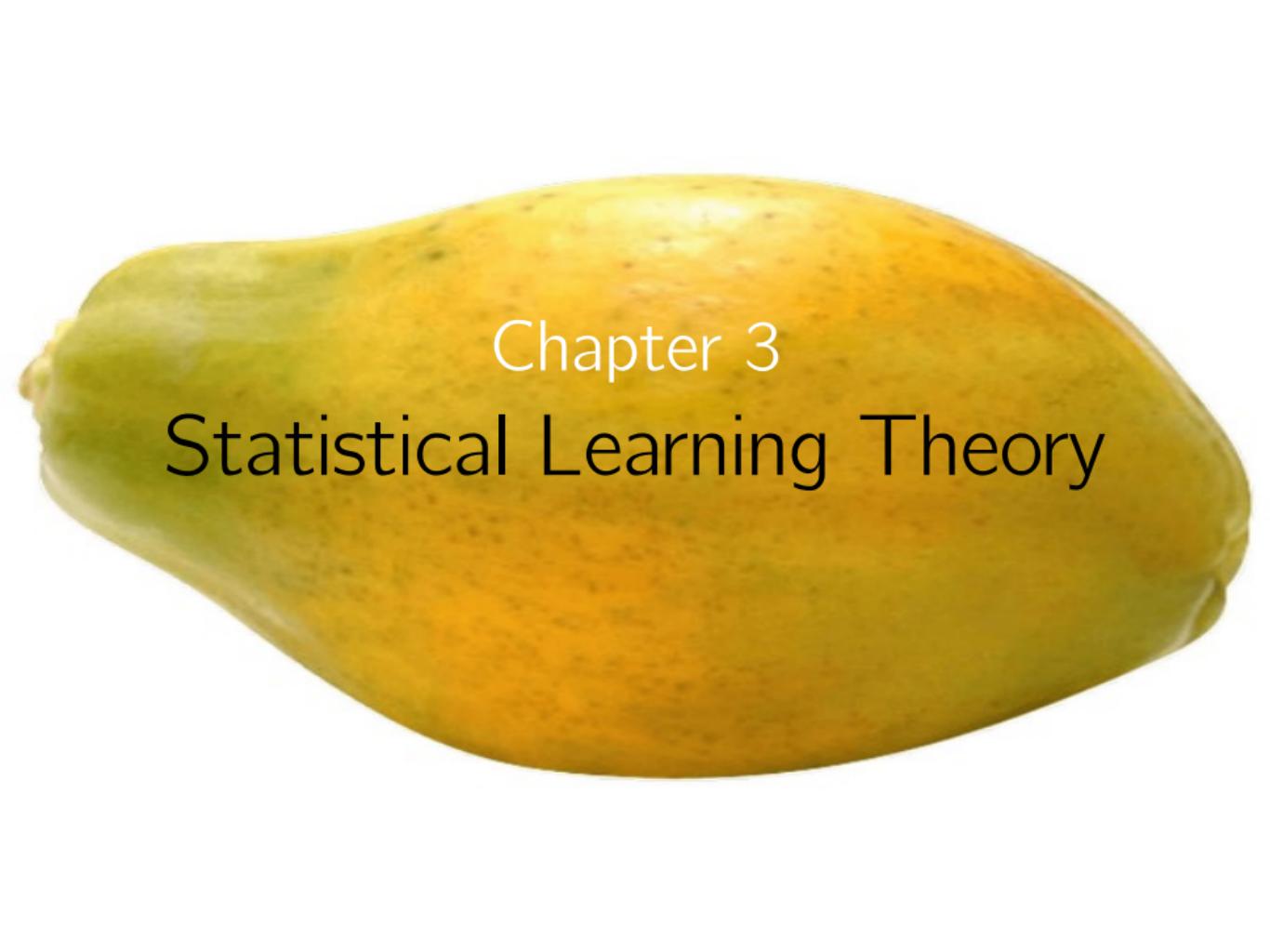
Remarks

- ▶ The Source Coding Theorem determines the limits of compression; it does not give a practical compression scheme.
- ▶ There is also a version of the Source Coding Theorem for so-called Symbol Codes. In this version, the codes are lossless, but there is only a probabilistic guarantee on the compression rate. Again, entropy is the best possible (expected) compression rate.
- ▶ Some of the theoretically optimal symbol codes, for example **Huffman Codes**, are also practically useful.
- ▶ However, all these results are proved under the i.i.d. assumption, which often does not hold in practical compression scenarios.
- ▶ For practical compression, we not only exploit skew in the distribution of the individual symbols, but also dependencies between successive symbols.
- ▶ For us, the main point of the Source Coding Theorem is that it formally confirms our intuition of a connection between entropy and compressibility.

References

Most of the material of this chapter can be found in [MacKay, 2003, Chapters 2 and 4].

To refresh your background in probability theory, [Wasserman, 2005, Chapters 1-4] may be useful.

A large, ripe yellow mango with a textured surface, centered against a white background.

Chapter 3

Statistical Learning Theory

Goals of this Chapter

Statistical learning theory aims at giving statistical estimates on the generalisation capabilities of a machine learning model. The goals of this chapter are twofold:

- ▶ understand the basic assumptions of statistical learning theory and the PAC learning framework;
- ▶ understand generalisation bounds relating the training error and the generalisation error of empirical risk minimisation algorithms.

3.1 Classification Revisited

Task

Learn to predict whether an unknown fruit, say, a papaya, is tasty or not (from observations we can make from the outside).

As a **training set**, we get a sample of papayas that we can try.



Remark 3.1

This is an example of a Boolean classification problem in a supervised learning setting.

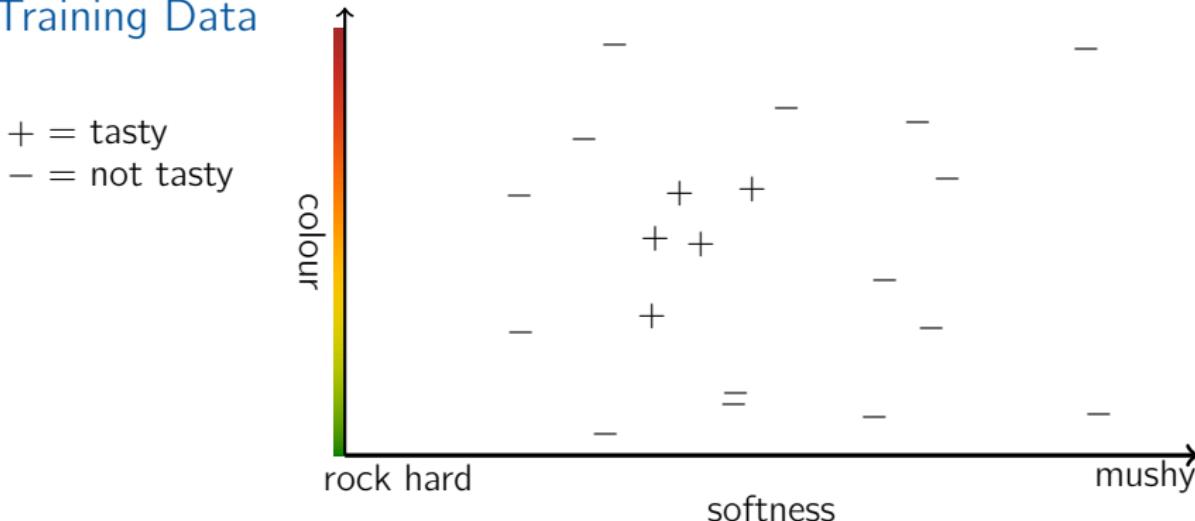
Example (cont'd)

Feature selection

Maybe based on experience with other fruit, we decide to base our decision on two **features** of the papayas:

- ▶ **colour**, ranging from green through yellow and red to brown
 - ▶ **softness**, ranging from hard to mushy

Training Data



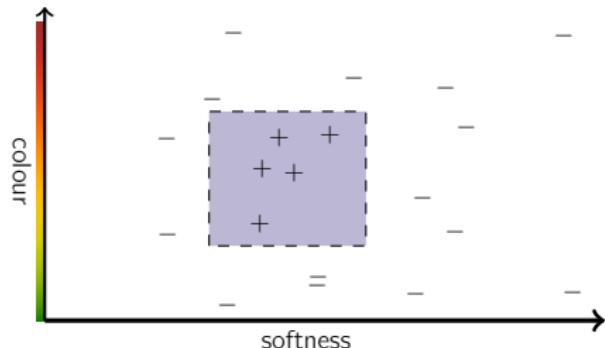
Example (cont'd)

Model selection

We may choose a simple **parametric model**: all papayas whose colour is in a certain range $[c_{\min}, c_{\max}]$ and whose softness is in a range $[s_{\min}, s_{\max}]$.

Parameter estimation

We try to estimate, or learn, the parameters $c_{\min}, c_{\max}, s_{\min}, s_{\max}$ in such a way that the resulting **hypothesis** explains the data best.



In our example, the **space of all possible hypotheses** is the set of all axis-parallel rectangles in the “color-softness plane”.

Example (cont'd)

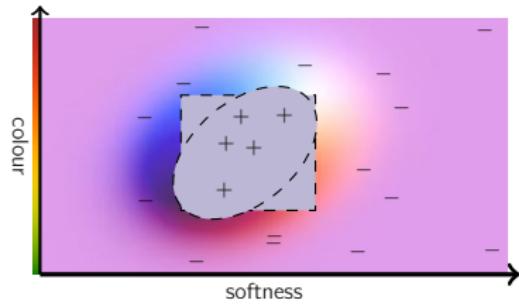
Model learning

We can also try to learn the model and the parameters for this model (if there are any) directly.

However, we have to be careful to avoid overfitting.

Remarks 3.2

- ▶ Our model (hypothesis space) of rectangles in the plane is extremely simple.
Even the feature selection and the mapping of features to real numbers are questionable.
- ▶ As our features probably won't determine the tastiness of a papaya alone, a more realistic type of model would just output a probability that the corresponding papayas are tasty for each point in the plane.



Goal

Given the training examples, we want to “learn” a hypothesis that **generalises well**, that is, is a good approximation of the unknown target function.

- ▶ So far, we have only discussed how to evaluate a learning algorithm experimentally.
- ▶ In the next section, we will introduce a mathematical framework that will allow us to formulate exactly what we expect of a good learning algorithm.
- ▶ It will also enable us to give estimates on how many training examples we need to see to obtain a reliable hypothesis.

3.2 The PAC Learning Framework

Goal

Given the training examples, we want to “learn” a hypothesis that **generalises well**, that is, is a good approximation of the unknown target function.

Caveat

The training data have to be “typical” data points representative for the whole instance space. We need to make assumptions guaranteeing this, otherwise we can’t expect to succeed.

I.i.d. Assumption

We assume that all data are drawn from an unknown probability distribution \mathcal{D} , the **data generating distribution**.

We assume that the training instances in the training set as well as future instances are **independent** and **identically distributed** according to this distribution \mathcal{D} .

Generalisation error

We want to minimise the expected error of our hypothesis on instances drawn from \mathcal{D} .

Formal Framework

We consider Boolean classification problems.

- ▶ We have an **instance space** \mathbb{X} , and our goal is to learn an unknown **target function**

$$f^* : \mathbb{X} \rightarrow \{0, 1\}.$$

- ▶ We assume furthermore that there is an unknown data generating probability distribution \mathcal{D} on \mathbb{X} .
- ▶ The **generalisation error**, or **risk**, of a hypothesis $h : \mathbb{X} \rightarrow \{0, 1\}$ is

$$\text{err}_{\mathcal{D}}(h) := \Pr_{x \sim \mathcal{D}} (h(x) \neq f^*(x))$$

Formal Framework (cont'd)

- Our algorithm receives as input a **training sequence**

$$T = \left((x_1, y_1), \dots, (x_m, y_m) \right) \in (\mathbb{X} \times \{0, 1\})^m,$$

where $y_i = f^*(x_i)$, and produces a **hypothesis** $h_T : \mathbb{X} \rightarrow \{0, 1\}$ from some **hypothesis class** \mathcal{H} .

- We write $T \sim \mathcal{D}^m$ to denote that the instances x_1, \dots, x_m of a training sequence T of length m are drawn independently from \mathcal{D} .
- The **training error**, or **empirical risk**, of a hypothesis $h : \mathbb{X} \rightarrow \{0, 1\}$ (w.r.t. the training sequence T) is

$$\text{err}_T(h) := \frac{1}{m} |\{i \in [m] \mid h(x_i) \neq y_i\}|.$$

If $\text{err}_T(h) = 0$ then h is **consistent** with T .

Probably Approximately Correct Learning

A learning algorithm that on input T produces a hypothesis h_T is a **PAC learning algorithm** if for all $\varepsilon, \delta > 0$ there is an $m = m(\varepsilon, \delta)$ such that for every probability distribution \mathcal{D} on \mathbb{X}

$$\Pr_{T \sim \mathcal{D}^m} (\text{err}_{\mathcal{D}}(h_T) \leq \varepsilon) > 1 - \delta.$$



Photo: CACM, Vol. 54, No. 6

Leslie Valiant (*1949)

ACM Turing Award 2010

For transformative contributions to the theory of computation, including the theory of probably approximately correct (PAC) learning, the complexity of enumeration and of algebraic computation, and the theory of parallel and distributed computing.

Empirical Risk Minimisation

Empirical Risk Minimisation

An **ERM algorithm** with a hypothesis class \mathcal{H} returns, on input T , a hypothesis h_T such that

$$\text{err}_T(h_T) = \min_{h \in \mathcal{H}} \text{err}_T(h).$$

Remarks 3.3

Recall that a learning problem is **realisable** if the target function is contained in the hypothesis space.

Note that if the problem is realisable, then an ERM algorithm always returns a consistent hypothesis.

Regularisation

ERM runs the risk of **overfitting**, in particular for hypothesis classes \mathcal{H} of high **capacity** (i.e., “rich” classes).

To counteract overfitting, we can add a **regularisation term** to the function we minimise:

$$h_T = \arg \min_{h \in \mathcal{H}} \left(\text{err}_T(h) + \rho(\text{cost}(h)) \right)$$

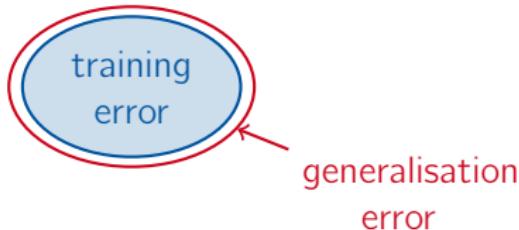
where:

- ▶ cost is a function that associates a “complexity cost” to every hypothesis.

Examples: bit precision of a real vector, degree of a polynomial, size of a decision tree

- ▶ ρ is an arbitrary monotone function, often just a linear function.

3.3 Sample Size Bounds



- ▶ A learning algorithm only sees the training error, but it actually wants to minimise the generalisation error.
- ▶ An ERM algorithm tries to achieve this by minimising the training error.
- ▶ Thus an ERM algorithm is good if the training error is close to the generalisation error.
- ▶ In this section, we shall prove bounds of the form: "*If we see sufficiently many examples then the generalisation error is close to the training error.*"

Realisability and Agnostic Learning

- ▶ Even a guarantee that **the training error is close to the generalisation error for all possible hypotheses** does not guarantee a hypothesis with a small generalisation error, simply because the hypothesis space may be badly chosen and there may not be a hypothesis with a small training/generalisation error.
- ▶ For realisable learning problems, there is always a hypothesis of training and generalisation error 0, but realisability is a strong assumption that we cannot always make. Instead, we will try to find the best possible hypothesis in our hypothesis space (regardless of whether the target function is in the space or not). Such an approach to learning that does not make a realisability assumption (but works particularly well if the problem happens to be realisable) is sometimes called **agnostic learning**.

Sample Size Bound

Theorem 3.4 (Simple Sample Size Bound)

Let \mathcal{H} be a finite hypothesis class. Let $\varepsilon, \delta > 0$ and

$$m \geq \frac{1}{\varepsilon} \ln \left(\frac{|\mathcal{H}|}{\delta} \right).$$

Then for any data generating distribution \mathcal{D} ,

$$\Pr_{T \sim \mathcal{D}^m} \left(\forall h \in \mathcal{H} : \text{if } h \text{ is consistent with } T, \text{ then } \text{err}_{\mathcal{D}}(h) \leq \varepsilon \right) > 1 - \delta.$$

Corollary 3.5

Suppose the hypothesis class is finite and it satisfies the realisability assumption.

Then an ERM algorithm that for $\varepsilon, \delta > 0$ sees $m \geq \frac{1}{\varepsilon} \ln \left(\frac{|\mathcal{H}|}{\delta} \right)$ examples is a PAC learning algorithm.

The proof of the sample size bound is a good occasion to recall another simple fact from probability theory.

Union Bound

For events A_1, \dots, A_k in some probability space,

$$\Pr\left(\bigcup_{i=1}^k A_i\right) \leq \sum_{i=1}^k \Pr(A_i).$$

We also need the following inequality, which holds for all $x \in \mathbb{R}$:

$$(1 + x) \leq e^x. \quad (\star)$$

Proof of the theorem.

Let h_1, \dots, h_n be the list of all hypotheses $h \in \mathcal{H}$ of generalisation error $\text{err}_{\mathcal{D}}(h) > \varepsilon$. Then $n \leq |\mathcal{H}|$ and thus

$$m \geq \frac{1}{\varepsilon} \ln\left(\frac{n}{\delta}\right). \quad (\star\star)$$

For each $i \in [n]$ we have

$$\Pr_{x \sim \mathcal{D}} (h_i(x) = f^*(x)) = 1 - \text{err}_{\mathcal{D}}(h_i) < 1 - \varepsilon,$$

where f^* denotes the target function. Thus

$$\Pr_{T \sim \mathcal{D}^m} (\text{err}_T(h_i) = 0) < (1 - \varepsilon)^m.$$

Hence by the Union Bound and by (\star) applied to $x = -\varepsilon$,

$$\Pr_{T \sim \mathcal{D}^m} (\exists i \in [n] : \text{err}_T(h_i) = 0) < n(1 - \varepsilon)^m \leq ne^{-\varepsilon m}.$$

Plugging in inequality $(\star\star)$ yields

$$\Pr_{T \sim \mathcal{D}^m} (\exists i \in [n] : \text{err}_T(h_i) = 0) < \delta$$

by a straightforward calculation.

By the choice of h_1, \dots, h_n , a hypothesis $h \in \mathcal{H}$ has generalisation error $\text{err}_{\mathcal{D}}(h) > \varepsilon$ iff $h = h_i$ for some $i \in [n]$. Thus

$$\Pr_{T \sim \mathcal{D}^m} (\exists h \in \mathcal{H} : \text{err}_T(h) = 0 \wedge \text{err}_{\mathcal{D}}(h) > \varepsilon)$$

$$= \Pr_{T \sim \mathcal{D}^m} (\exists i \in [n] : \text{err}_T(h_i) = 0) \\ < \delta.$$

By the simple logical equivalence

$$\neg \exists x (A(x) \wedge B(x)) \iff \forall x (A(x) \implies \neg B(x)),$$

this yields the assertion of the theorem:

$$\begin{aligned} & \Pr_{T \sim \mathcal{D}^m} (\forall h \in \mathcal{H} : \text{err}_T(h) = 0 \implies \text{err}_{\mathcal{D}}(h) \leq \varepsilon) \\ &= \Pr_{T \sim \mathcal{D}^m} (\neg \exists h \in \mathcal{H} : \text{err}_T(h) = 0 \wedge \text{err}_{\mathcal{D}}(h) > \varepsilon) \\ &> 1 - \delta. \end{aligned}$$

□

Proof of the corollary.

If the learning problem is realisable, on input T an ERM algorithm produces a hypothesis $h_T \in \mathcal{H}$ consistent with T . By the theorem, with probability $> 1 - \delta$, every consistent hypothesis $h \in \mathcal{H}$ has a generalisation error $\text{err}_{\mathcal{D}}(h) \leq \varepsilon$, and thus in particular $\text{err}_{\mathcal{D}}(h_T) \leq \varepsilon$.

□

Example (Discretisation Trick)

Suppose that our hypothesis space consists of linear separators in 10-dimensional Euclidean space that our learning algorithm specifies by giving a normal vector \mathbf{w} and a bias b in 64 bit floating point numbers. For example, the learning algorithm may be the perceptron.

The size of the hypothesis space of the algorithm is $2^{11 \cdot 64} = 2^{704}$.

We let $\varepsilon := 0.1$ and $\delta := 0.05$. If

- ▶ our algorithm receives a training sequence of size (at least)

$$\frac{1}{\varepsilon} \ln \left(\frac{|\mathcal{H}|}{\delta} \right) = 10 \ln (20 \cdot 2^{704}) \approx 4910,$$

- ▶ and produces a hypothesis (a linear separator) consistent with the training data,

then with probability larger than 95% the generalisation error is at most 10%.

Uniform Convergence

Theorem 3.6 (Uniform Convergence)

Let \mathcal{H} be a finite hypothesis class. Let $\varepsilon, \delta > 0$ and

$$m \geq \frac{1}{2\varepsilon^2} \ln \left(\frac{2|\mathcal{H}|}{\delta} \right).$$

Then for any data generating distribution \mathcal{D} ,

$$\Pr_{T \sim \mathcal{D}^m} \left(\forall h \in \mathcal{H} : |\text{err}_T(h) - \text{err}_{\mathcal{D}}(h)| \leq \varepsilon \right) > 1 - \delta.$$

Proof.

Always suppose that $T = ((x_1, y_1), \dots, (x_m, y_m))$.

Let $h \in \mathcal{H}$. Let X_i be the indicator random variable for the event that $h(x_i) \neq y_i$ and $X := \sum_{i=1}^m X_i$. Then

$$\Pr(X_i = 1) = \text{err}_{\mathcal{D}}(h)$$

and thus $\mathbb{E}(X) = m \text{err}_{\mathcal{D}}(h)$. Moreover,

$$\frac{X}{m} = \text{err}_T(h).$$

Thus by the Hoeffding Bounds (Corollary 2.7),

$$\begin{aligned} \Pr_{T \sim \mathcal{D}^m} (|\text{err}_T(h) - \text{err}_{\mathcal{D}}(h)| > \varepsilon) &= \Pr \left(\left| \frac{X}{m} - \frac{\mathbb{E}(X)}{m} \right| > \varepsilon \right) \\ &= \Pr (|X - \mathbb{E}(X)| > \varepsilon m) \\ &< 2e^{-2m\varepsilon^2}. \end{aligned}$$

By the Union Bound,

$$\Pr_{T \sim \mathcal{D}^m} (\exists h \in \mathcal{H} : |\text{err}_T(h) - \text{err}_{\mathcal{D}}(h)| \geq \varepsilon) < 2|\mathcal{H}|e^{-2m\varepsilon^2}.$$

To complete the proof of the theorem, we show that $2|\mathcal{H}|e^{-2m\varepsilon^2} \leq \delta$:

$$\begin{aligned} 2|\mathcal{H}|e^{-2m\varepsilon^2} &\leq 2|\mathcal{H}|2^{-2\frac{1}{2\varepsilon^2} \ln\left(\frac{2|\mathcal{H}|}{\delta}\right)\varepsilon^2} \quad \text{because } m \geq \frac{1}{2\varepsilon^2} \ln\left(\frac{2|\mathcal{H}|}{\delta}\right) \\ &= 2|\mathcal{H}|2^{-\ln\left(\frac{2|\mathcal{H}|}{\delta}\right)} \\ &= 2|\mathcal{H}|\frac{1}{\frac{2|\mathcal{H}|}{\delta}} \\ &= \delta. \end{aligned}$$

□

Agnostic PAC Learning

Theorem 3.7 (Agnostic PAC Learning Sample Size Bound)

Consider an ERM algorithm with a finite hypothesis class \mathcal{H} that given T produces a hypothesis h_T .

Let $\varepsilon, \delta > 0$ and

$$m \geq \frac{2}{\varepsilon^2} \ln \left(\frac{2|\mathcal{H}|}{\delta} \right).$$

Let \mathcal{D} be a data generating distribution and $h^* = \arg \min_{h \in \mathcal{H}} \text{err}_{\mathcal{D}}(h)$.

Then

$$\Pr_{T \sim \mathcal{D}^m} \left(|\text{err}_{\mathcal{D}}(h_T) - \text{err}_{\mathcal{D}}(h^*)| \leq \varepsilon \right) > 1 - \delta.$$

Proof.

By the optimality of h^* we have

$$\text{err}_{\mathcal{D}}(h^*) \leq \text{err}_{\mathcal{D}}(h_T). \quad (\star)$$

As an ERM algorithm minimises the training error, for all T we have

$$\text{err}_T(h_T) \leq \text{err}_T(h^*). \quad (\star\star)$$

By the Uniform Convergence Theorem, with probability $> 1 - \delta$ we have both

$$|\text{err}_T(h^*) - \text{err}_{\mathcal{D}}(h^*)| \leq \frac{\varepsilon}{2} \quad \text{and} \quad |\text{err}_T(h_T) - \text{err}_{\mathcal{D}}(h_T)| \leq \frac{\varepsilon}{2}. \quad (\heartsuit)$$

Thus, with probability $> 1 - \delta$,

$$\begin{aligned} |\text{err}_{\mathcal{D}}(h_T) - \text{err}_{\mathcal{D}}(h^*)| &= \text{err}_{\mathcal{D}}(h_T) - \text{err}_{\mathcal{D}}(h^*) && \text{by } (\star) \\ &= \text{err}_{\mathcal{D}}(h_T) - \text{err}_T(h_T) \\ &\quad + \text{err}_T(h_T) - \text{err}_T(h^*) \end{aligned}$$

$$\begin{aligned}
& + \text{err}_T(h^*) - \text{err}_{\mathcal{D}}(h^*) \\
& \leq \text{err}_{\mathcal{D}}(h_T) - \text{err}_T(h_T) \\
& \quad + \text{err}_T(h^*) - \text{err}_{\mathcal{D}}(h^*) \quad \text{by } (\star\star) \\
& \leq \varepsilon \quad \text{by } (\heartsuit).
\end{aligned}$$

□

All the results proved so far in this chapter only apply to finite hypothesis classes. We will now discuss generalisations to infinite classes.

Occam's Razor

Choose the simplest hypothesis consistent with the data.

We will prove that this paradigm guarantees good results, provided we have enough training data.

Description Scheme

Let \mathcal{H} be an arbitrary, possibly infinite, class of hypotheses.

- ▶ Assume that we have some scheme Δ for describing the hypotheses $h \in \mathcal{H}$ by strings in Σ^* for some finite alphabet Σ (of size $|\Sigma| \geq 2$).
- ▶ For every $h \in \mathcal{H}$, we let $|h|_\Delta$ be the length of the shortest description of h .

Now we can measure the “simplicity” of a hypothesis h as the description length $|h|_\Delta$.

Theorem 3.8

Let $n \in \mathbb{N}$, $\varepsilon, \delta > 0$, and

$$m \geq \frac{1}{\varepsilon} \left(n \ln |\Sigma| + \ln \left(\frac{2}{\delta} \right) \right).$$

Then for any data generating distribution \mathcal{D} ,

$$\Pr_{T \sim \mathcal{D}^m} \left(\forall h \in \mathcal{H} : (|h|_\Delta \leq n \wedge \text{err}_T(h) = 0 \implies \text{err}_{\mathcal{D}}(h) \leq \varepsilon) \right) > 1 - \delta.$$

Proof.

We apply the Simple Sample Size Bound (Theorem 3.4) to the hypothesis class

$$\mathcal{H}_n := \{h \in \mathcal{H} \mid |h|_{\Delta} \leq n\}.$$

We have

$$|\mathcal{H}_n| \leq \sum_{i=0}^n |\Sigma|^i = \frac{|\Sigma|^{n+1} - 1}{|\Sigma| - 1} \leq 2|\Sigma|^n$$

(assuming $|\Sigma| \geq 2$) and thus

$$\ln \left(\frac{|\mathcal{H}_n|}{\delta} \right) \leq \ln \left(\frac{2|\Sigma|^n}{\delta} \right) = n \ln |\Sigma| + \ln \left(\frac{2}{\delta} \right).$$



- ▶ The theorem suggests that if we always choose the simplest hypothesis consistent with the data, then we can't go wrong, provided we have enough data to support the hypothesis.
- ▶ The remarkable thing is that this does not depend on the description scheme (and hence the measure of "simplicity").
- ▶ Note that the theorem does not say that the simplest hypothesis is the best hypothesis, it allows for more complicated hypotheses to be even better. The theorem just says that the simplest one can never be bad.

Application to Decision Trees

- ▶ Let ℓ be the number of features (that is, the dimension of the instance space).
- ▶ A decision tree with n nodes can be described using $O(n(\log n + \log \ell))$ bits.
- ▶ Hence if we have a decision tree with n nodes consistent with a randomly chosen training set of size $\frac{1}{\epsilon}(cn \ln n + \ln(1/\delta))$ for some constant c not depending on n , then with probability $> (1 - \delta)$ the hypothesis described by the tree has error at most ϵ .

VC Dimension

There is another way of generalising our sample size bounds to infinite hypothesis classes. For this, rather than the size, we try to quantify the “complexity” of the hypothesis class \mathcal{H} .

VC dimension is a combinatorial measure for the complexity of a class of Boolean functions.

Let \mathcal{H} be a class of functions $h : \mathbb{X} \rightarrow \{0, 1\}$.

- ▶ A subset $Y \subseteq \mathbb{X}$ is shattered by \mathcal{H} if every function $g : Y \rightarrow \{0, 1\}$ is the restriction of a function in \mathcal{H} to Y .
- ▶ The VC-dimension (Vapnik-Chervonenkis dimension) $VC(\mathcal{H})$ of \mathcal{H} is the size of the largest set shattered by \mathcal{H} , or ∞ if arbitrarily large sets are shattered by \mathcal{H} .

Example

Let $\mathbb{X} = \mathbb{R}^2$ and \mathcal{H} the class of all axis-parallel rectangles. Then $\text{VC}(\mathcal{H}) = 4$.

Proof.

To see that $\text{VC}(\mathcal{H}) \geq 4$, note that the four corners of a diamond can be shattered by rectangles.

To see that $\text{VC}(\mathcal{H}) < 5$, let $Y \subseteq \mathbb{X}$ such that $|Y| = 5$. Let $H \in \mathcal{H}$ be a rectangle that encloses Y , that is, a minimal rectangle with $Y \subseteq H$. Then there is a subset $Z \subset Y$ of at most four points such that each side of H contains one element of Z . Every rectangle $H' \in \mathcal{H}$ that contains Z must also contain H and thus Y , which implies that Y is not shattered by \mathcal{H} . \square

Example

Let $\mathbb{X} = \mathbb{R}$ and \mathcal{H} the class of all finite subsets of \mathbb{X} . Then $\text{VC}(\mathcal{H}) = \infty$.

Uniform Convergence for VC Dimension

Theorem 3.9

Let \mathcal{H} be a hypothesis class of finite VC-dimension d .

Let $\varepsilon, \delta > 0$ and

$$m \geq \frac{c}{\varepsilon^2} \left(d + \ln \left(\frac{1}{\delta} \right) \right)$$

(for a suitable constant c).

Then for any data generating distribution \mathcal{D} ,

$$\Pr_{T \sim \mathcal{D}^m} \left(\forall h \in \mathcal{H} : |\text{err}_T(h) - \text{err}_{\mathcal{D}}(h)| \leq \varepsilon \right) > 1 - \delta.$$

(Without proof.)

References

The material of this chapter can be found in
[Shalev-Shwartz and Ben-David, 2014, Chapters 2–6] and also in
[Blum et al., 2020, Chapter 6].

Chapter 4

Multiplicative Weight Updates



Goals of this Chapter

Multiplicative weight updates is a versatile algorithmic idea with many applications. The goal of this chapter are:

- ▶ understand the basic algorithm and its analysis using suitable potential functions;
- ▶ see how it can adapted to different scenarios.

4.1 The MWU Algorithm

Setting

- ▶ We invest in a certain stock. Every day, we either invest 1€ or we don't invest.
- ▶ **Simplifying assumption:** Price movements are binary events: up/down.
If we invest and the price goes down or we don't invest and the price goes up, we loose 1€.
- ▶ We try to minimise losses (over time).
- ▶ In making our investment, we are allowed to take the advice of several “experts”. Over time, we see the past performance of these experts.

Remark 4.1

We only know in retrospect who is the best expert. Still, our algorithm will perform almost as good as the best expert.

We assume that we have n experts numbered $1, \dots, n$.

For every $t \geq 1$:

- ▶ $p^{(t)} \in \{0, 1\}$: price movement on day t ($0 = \text{down}$, $1 = \text{up}$),
- ▶ $a_i^{(t)} \in \{0, 1\}$ for $i \in [n]$: advice of expert i for day t ($0 = \text{don't buy}$, $1 = \text{buy}$),
- ▶ $\ell_i^{(t)} := \sum_{s=1}^t |a_i^{(s)} - p^{(s)}|$ for $i \in [n]$: cumulated loss of expert i after t days,
- ▶ $d^{(t)} \in \{0, 1\}$: our decision on day t ($0 = \text{don't buy}$, $1 = \text{buy}$),
- ▶ $\ell^{(t)} := \sum_{s=1}^t |d^{(s)} - p^{(s)}|$: our cumulated loss after t days.
- ▶ In addition, our algorithm will assign a weight $w_i^{(t)}$ to every expert on every day.

Weighted Majority Algorithm

For some constant $0 < \alpha \leq 1/2$, we assign weights as follows.

- ▶ $w_i^{(1)} := 1$ for all $i \in [n]$.

Intuition: Initially, we give the same weight to each expert's advice.

- ▶ For $t \geq 1$, $d^{(t)} := \begin{cases} 1 & \text{if } \sum_{i \in [n]} w_i^{(t)} \geq \sum_{i \in [n]} a_i^{(t)} \\ 0 & \text{otherwise.} \end{cases}$

Intuition: Buy if the weighted majority of the experts recommends it.

- ▶ For $t \geq 1$ and $i \in [n]$, $w_i^{(t+1)} := \begin{cases} w_i^{(t)} & \text{if } a_i^{(t)} = p^{(t)}, \\ (1 - \alpha)w_i^{(t)} & \text{otherwise} \end{cases}$

Intuition: Decrease weights of experts with wrong prediction by a factor $(1 - \alpha)$.

Theorem 4.2

For every $t \geq 1$ and every $i \in [n]$,

$$\ell^{(t)} \leq \frac{2 \ln n}{\alpha} + 2(1 + \alpha)\ell_i^{(t)}.$$

Intuition: The inequality holds, in particular, for i being the best expert. Thus in the long run, our algorithm guarantees our losses to be just a bit more than twice the losses of the best expert.

Proof of the Theorem.

It will be convenient to let $\ell_i^{(0)} := 0$ for all i .

Claim 1

For all $t \geq 0$ and $i \in [n]$ we have $w_i^{(t+1)} = (1 - \alpha)^{\ell_i^{(t)}}$.

Proof of the Claim: Induction on t . □

Potential function:

$$\Phi^{(t)} := \sum_{i=1}^n w_i^{(t)}.$$

Thus by Claim 1,

$$\Phi^{(t+1)} = \sum_{i=1}^n (1 - \alpha)^{\ell_i^{(t)}}. \quad (\star)$$

Observe:

- ▶ $\Phi^{(1)} = n$.
- ▶ $\Phi^{(t+1)} \leq \Phi^{(t)}$.

- If $d^{(t)} \neq p^{(t)}$, then

$$\sum_{\substack{i \in [n] \\ a_i^{(t)} \neq p^{(t)}}} w_i^{(t)} \geq \sum_{\substack{i \in [n] \\ a_i^{(t)} = p^{(t)}}} w_i^{(t)},$$

which implies

$$\sum_{\substack{i \in [n] \\ a_i^{(t)} \neq p^{(t)}}} w_i^{(t)} \geq \frac{\sum_{i \in [n]} w_i^{(t)}}{2} = \frac{\Phi^{(t)}}{2}.$$

Thus (still if $d^{(t)} \neq p^{(t)}$)

$$\begin{aligned} \Phi^{(t+1)} &= (1 - \alpha) \sum_{\substack{i \in [n] \\ a_i^{(t)} \neq p^{(t)}}} w_i^{(t)} + \sum_{\substack{i \in [n] \\ a_i^{(t)} = p^{(t)}}} w_i^{(t)} \\ &= \sum_{i \in [n]} w_i^{(t)} - \alpha \sum_{\substack{i \in [n] \\ a_i^{(t)} \neq p^{(t)}}} w_i^{(t)} \\ &\leq \Phi^{(t)} (1 - \alpha/2). \end{aligned}$$

Now an easy induction on t shows

$$\Phi^{(t+1)} \leq n(1 - \alpha/2)^{\ell^{(t)}}. \quad (\star\star)$$

Comparing (\star) and $(\star\star)$, for all $t \geq 1$ we get

$$\sum_{i=1}^n (1 - \alpha)^{\ell_i^{(t)}} \leq n(1 - \alpha/2)^{\ell^{(t)}}.$$

Thus for all $i \in [n]$,

$$\begin{aligned} (1 - \alpha)^{\ell_i^{(t)}} &\leq n(1 - \alpha/2)^{\ell^{(t)}} \\ \implies \ell_i^{(t)} \ln(1 - \alpha) &\leq \ln n + \ell^{(t)} \ln(1 - \alpha/2) \\ \implies \frac{\ell_i^{(t)} \ln(1 - \alpha) - \ln n}{\ln(1 - \alpha/2)} &\geq \ell^{(t)}. \end{aligned}$$

Now the assertion of the lemma follows from the following two claims.

Claim 2

$$-\ln(1 - \alpha/2) \geq \alpha/2$$

and hence

$$-\frac{1}{\ln(1 - \alpha/2)} \leq \frac{2}{\alpha}.$$

Proof of the claim: We use the inequality $(1 + x) \leq e^x$, which holds for all $x \in \mathbb{R}$. It yields

$$\begin{aligned}\implies \quad & \left(1 - \frac{\alpha}{2}\right) \leq e^{-\frac{\alpha}{2}} \\ \implies \quad & \ln\left(1 - \frac{\alpha}{2}\right) \leq -\frac{\alpha}{2} \\ \implies \quad & -\ln\left(1 - \frac{\alpha}{2}\right) \geq \frac{\alpha}{2}.\end{aligned}$$

□

Claim 3

$$\frac{\ln(1 - \alpha)}{\ln(1 - \alpha/2)} \leq 2(1 + \alpha)$$

Proof of the claim: By Claim 2 we have

$$-\ln(1 - \alpha/2) \geq \alpha/2.$$

The Taylor expansion

$$\ln(1 - x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \dots,$$

which converges for $-1 < x < 1$, yields the inequality $\ln(1 - x) \geq -x - x^2$ for $0 \leq x \leq 1/2$. Thus

$$-\ln(1 - \alpha) \leq \alpha + \alpha^2 = \alpha(1 + \alpha).$$

Thus

$$\begin{aligned}\frac{\ln(1 - \alpha)}{\ln(1 - \alpha/2)} &= \frac{-\ln(1 - \alpha)}{-\ln(1 - \alpha/2)} \\ &\leq \frac{\alpha(1 + \alpha)}{\frac{\alpha}{2}} \\ &= 2(1 + \alpha).\end{aligned}$$



- ▶ Again, we have a set I of n “experts”; usually $I = [n]$.
- ▶ We have a finite set J of possible “events”.
- ▶ We have a “loss matrix” $L \in \mathbb{R}^{I \times J}$ with nonnegative entries, where the entry L_{ij} describes our loss when following expert i while event j happened.

We assume the matrix to be normalised so that all entries are in the closed interval $[0, 1]$.

- ▶ Again, we play in discrete timesteps $t \geq 1$ (we call them “rounds”).
- ▶ This time, we invest using a randomised strategy: in each round t we draw an expert i randomly from some probability distribution $\mathcal{D}^{(t)}$. We follow this expert i , and thus if event j happens, our loss is L_{ij} .

Our goal is to minimise the expected loss.

Notation

- ▶ $j^{(t)} \in J$: event that happened at time t
- ▶ Our algorithm assigns the weight $w_i^{(t)}$ to expert i in round t .
- ▶ The probability distribution $\mathcal{D}^{(t)}$ on I in round t is defined by

$$\mathcal{D}^{(t)}(\{i\}) := p_i^{(t)} := \frac{w_i^{(t)}}{\sum_{i' \in I} w_{i'}^{(t)}}$$

- ▶ It will be our strategy to follow the advice of expert i in round t with probability $p_i^{(t)}$.
- ▶ Then our expected loss in round t is

$$L^{(t)} := \sum_{i \in I} p_i^{(t)} L_{ij^{(t)}}.$$

Multiplicative Weight Update (MWU) Algorithm

For some constant $0 < \alpha < 1$.

- ▶ $w_i^{(1)} := 1$ for all $i \in I$.
- ▶ For $t \geq 1$ and $i \in I$,

$$w_i^{(t+1)} := (1 - \alpha)^{L_{ij(t)}} w_i^{(t)}.$$

Theorem 4.3

For every $t \geq 1$ and every $i \in I$,

$$\sum_{s=1}^t L^{(s)} \leq \frac{\ln n}{\alpha} + (1 + \alpha) \sum_{s=1}^t L_{ij^{(s)}}$$

(Proof omitted.)

- ▶ By choosing α to be small we can bring the multiplicative error factor arbitrarily close to 1. We pay for this with a larger additive error term, which is usually most significant for small t .
- ▶ In the simplified setting where we considered the weighted majority algorithm, the randomised strategy of the multiplicative weight update algorithm beats the deterministic strategy of the weighted majority algorithm by almost a factor of 2 (in expectation).

The multiplicative weight update method has numerous applications in different areas:

- ▶ Machine learning: **boosting** and **bandit learning**
- ▶ Game theory and economics (“no regret learning”)
- ▶ Design of approximation algorithms
- ▶ Computational complexity
- ▶ Computational evolutionary biology

4.2 Boosting Weak Learning Algorithms

Review: Classification Problems

- ▶ We consider Boolean classification problems, where we want to learn an unknown target function $f^* : \mathbb{X} \rightarrow \{0, 1\}$ on an instance space \mathbb{X} .
- ▶ A learning algorithm learns from a training sequence

$$T = ((x_1, y_1), \dots, (x_m, y_m))$$

of examples (x_i, y_i) , where

- ▶ the x_i are drawn independently from a data-generating probability distribution \mathcal{D} on \mathbb{X}
- ▶ $y_i = f^*(x_i)$,

and produces a hypothesis h from a hypothesis class \mathcal{H} .

- ▶ The generalisation error of h is

$$\text{err}_{\mathcal{D}}(h) := \Pr_{x \sim \mathcal{D}} (h(x) \neq f^*(x)).$$

- ▶ The training error of h on T is

$$\text{err}_T(h) := \frac{1}{m} |\{i \in [m] \mid h(x_i) \neq y_i\}|.$$

Weak and Strong Learner

Strong learner

Recall that a learning algorithm that on input T produces a hypothesis h_T is a **PAC learning algorithm**, or **strong learner**, if for all $\varepsilon, \delta > 0$ there is an $m = m(\varepsilon, \delta)$ such that for every probability distribution \mathcal{D} on \mathbb{X}

$$\Pr_{T \sim \mathcal{D}^m} (\text{err}_{\mathcal{D}}(h_T) \leq \varepsilon) > 1 - \delta.$$

Weak learner

Let $0 \leq \gamma < 1/2$. A learning algorithm that on input T produces a hypothesis h_T is a **weak learning algorithm with error parameter γ** (short: **γ -weak learner**) if for all $\delta > 0$ there is an $m = m(\delta)$ such that for every probability distribution \mathcal{D} on \mathbb{X}

$$\Pr_{T \sim \mathcal{D}^m} (\text{err}_{\mathcal{D}}(h_T) \leq \gamma) > 1 - \delta.$$

Boosting (Ideas)

Goal

Reduce the error of a weak learner; ultimately turning it into a strong learner.

Idea

- ▶ Repeatedly run the weak learner on subsets of the initial training set.
- ▶ These subsets are randomly drawn from different probability distributions.
- ▶ The distributions are adapted in each round using multiplicative weight updates.

Remark 4.4

This boosting algorithm we introduce here is called AdaBoost. This name reflects the adaptive updates of the distribution.

Boosting (Formal Setting)

Suppose we have a γ -weak learner \mathfrak{L} for our classification problem.

Boosting Problem

- Input:** Sufficiently long training sequence
 $T = ((x_1, y_1), \dots, (x_n, y_n))$ and error parameter $\varepsilon > 0$.
- Output:** Hypothesis h with $\text{err}_T(h) < \varepsilon$.

Remarks 4.5

- ▶ By choosing $\varepsilon < 1/n$, we can force h to be consistent with the training sequence.
- ▶ We don't insist on h coming from our original hypothesis class \mathcal{H} , but we want h to be from a class \mathcal{H}^* whose capacity is not much larger.
- ▶ Assuming that the training sequence is long enough (depending on the hypothesis class \mathcal{H}^* of our boosting algorithm), we can apply the sample size bounds of Chapter 3 to obtain a strong learner.

- ▶ Apply the weak learner \mathcal{L} repeatedly to examples drawn from different probability distributions $\mathcal{D}^{(t)}$ on

$$X := \{x_1, \dots, x_n\}.$$

- ▶ Adapt the probability distributions by putting less weight on examples that have previously been classified correctly.
- ▶ Use multiplicative weight updates to adapt the weights.

Setup for the Weak Learner \mathcal{L}

- ▶ We run \mathcal{L} with confidence parameter $\delta_0 = 1/10$ and let $m_0 = m(\delta_0) \leq n$ be the number of examples it needs.
- ▶ We identify probability distributions \mathcal{D}_X on X with probability distributions \mathcal{D} on \mathbb{X} by setting $\mathcal{D}(E) := \mathcal{D}_X(E \cap X)$ for all events $E \subseteq \mathbb{X}$.
In the following, we drop the notational distinction between \mathcal{D}_X and \mathcal{D} and denote both by \mathcal{D} .
- ▶ We only run \mathcal{L} with examples drawn according to probability distributions \mathcal{D} on X that we know.
- ▶ Call a hypothesis **good** if it has generalisation error less than γ . The weak learner \mathcal{L} generates a good hypothesis with probability at least $9/10 = 1 - \delta_0$.
- ▶ As we know \mathcal{D} and the correct labels for samples from X , we can check if a hypothesis is good.
- ▶ If \mathcal{L} generates a bad hypothesis, we repeatedly re-run it on new examples until it returns a good one.

With extremely high probability, this only requires few re-runs.

We say that we **run \mathcal{L} on \mathcal{D} until it returns a good hypothesis**.

Setup for the MWU Algorithm

- ▶ The set of “experts” is $I := [n]$.
- ▶ The set J of “events” is the set of all hypotheses generated by \mathcal{L} when presented with m_0 input examples from X .
- ▶ The loss matrix is defined by

$$L_{ij} := \begin{cases} 1 & \text{if } j(x_i) = y_i, \\ 0 & \text{otherwise.} \end{cases}$$

Note: The loss for i is positive and the weight will be decreased if a hypothesis is correct on x_i .

- ▶ We use the update parameter $\alpha = \frac{1}{2} - \gamma$.

Boosting Algorithm

- ▶ We consider a run of the MWU algorithm where $j^{(s)}$ is a hypothesis obtained by running \mathfrak{L} on $\mathcal{D}^{(s)}$ until it returns a good hypothesis.
- ▶ We run the algorithm for $t := \frac{2}{\alpha^2} \ln \frac{1}{\varepsilon}$ rounds.
- ▶ The final hypothesis h that we return is defined by

$$h(x) := \begin{cases} 1 & \text{if } |\{s \leq t \mid j^{(s)}(x) = 1\}| \geq t/2, \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 4.6

$$\text{err}_T(h) < \varepsilon.$$

Proof.

Let

$$E := \{i \in [n] \mid h(x_i) \neq y_i\}$$

be the set of examples misclassified by h , and let $\ell := |E|$.

Again we consider the potential function

$$\Phi^{(s)} := \sum_{i \in [n]} w_i^{(s)}.$$

Claim 1

$$\Phi^{(t)} \geq \ell(1 - \alpha)^{t/2} + (n - \ell)(1 - \alpha)^t.$$

Proof of Claim 1: An example misclassified by h is misclassified by at least $t/2$ of the hypotheses $j^{(s)}$. Thus its weight has been decreased at most $t/2$ times by a factor of $(1 - \alpha)$. \square

Claim 2

$$\Phi^{(t)} \leq n(1 - \alpha/2 - \alpha^2)^t.$$

Proof of Claim 2: We have $\Phi^{(1)} = n$, and for all $s \geq 1$ we have

$$\begin{aligned}\Phi^{(s+1)} &= (1 - \alpha) \sum_{\substack{i \in [n] \\ j^{(s)}(x_i) = y_i}} w_i^{(s)} + \sum_{\substack{i \in [n] \\ j^{(s)}(x_i) \neq y_i}} w_i^{(s)} \\ &= \Phi^{(s)} - \alpha \sum_{\substack{i \in [n] \\ j^{(s)}(x_i) = y_i}} w_i^{(s)}.\end{aligned}$$

By the property of the weak learning algorithm, we have

$$\begin{aligned}\gamma &> \text{err}_{\mathcal{D}^{(s)}}(j^{(s)}) \\ &= \Pr_{x \sim \mathcal{D}^{(s)}}(j^{(s)}(x) \neq y) \quad y = \text{value of target function on } x \\ &= \sum_{\substack{i \in [n] \\ j^{(s)}(x_i) \neq y_i}} p_i^{(s)}\end{aligned}$$

$$= \frac{\sum_{\substack{i \in [n] \\ j^{(s)}(x_i) \neq y_i}} w_i^{(s)}}{\Phi^{(s)}}.$$

Thus

$$\sum_{\substack{i \in [n] \\ j^{(s)}(x_i) = y_i}} w_i^{(s)} \geq (1 - \gamma) \Phi^{(s)}.$$

It follows that

$$\begin{aligned} \Phi^{(s+1)} &\leq \Phi^{(s)}(1 - \alpha(1 - \gamma)) \\ &= \Phi^{(s)}(1 - \alpha(1/2 + \alpha)) \quad \text{because } \alpha = 1/2 - \gamma \\ &= \Phi^{(s)}(1 - \alpha/2 - \alpha^2). \end{aligned}$$

The assertion of the claim follows by an easy induction. \square

Combining the two claims, we get

$$\ell(1 - \alpha)^{t/2} + (n - \ell)(1 - \alpha)^t \leq n(1 - \alpha/2 - \alpha^2)^t.$$

Now the result follows by a tedious, but straightforward calculation. \square

Running Time

- ▶ If we want our final hypothesis to classify all examples correctly, we need to take $\varepsilon \approx 1/n$ and thus run the MWU algorithm for $O(\log n)$ rounds.
- ▶ The running time is dominated by the calls to the weak learning algorithm \mathcal{L} . If \mathcal{L} is efficient, the boosting procedure is efficient as well.

4.3 Bandit Learning

Playing Against Adversarial Bandits



Informal Problem Description

- ▶ We repeatedly play one of n possible slot machines (“one-armed bandits”).
- ▶ In each round we choose one of the machines and observe the payoff (of the machine we pick, but not of the other machines). We can play a randomised strategy.
- ▶ Our goal is to minimise the expected **regret**, that is, the difference between the cumulative payoffs of our strategy and the best machine.
- ▶ We assume that the setting is **adversarial**, that is, an adversary fixes the payoff for each machine in each round in a way that maximises our regret.
The adversary knows our strategy in advance, but doesn't know the outcome of our random choices.

The bandit problem can be viewed as a simple example of a **reinforcement learning** scenario. However, the adversarial setting is more typical for economic scenarios.

Exploration vs Exploitation

The problem is a paradigmatic example of the trade-off between exploration and exploitation. On the one hand, if the gambler plays exclusively on the machine that he thinks is best ("exploitation"), he may fail to discover that one of the other arms actually has a higher expected payoff. On the other hand, if he spends too much time trying out all the machines and gathering statistics ("exploration"), he may fail to play the best arm often enough to get a high return.

P. Auer, N. Cesa-Bianchi, Y. Freund, R. Schapire: *The nonstochastic bandit problem*. SIAM J. Comput. 32, 2002.

Formal Problem Description

- ▶ We have n **actions** (the slot machines) numbered $1, \dots, n$.
- ▶ For every $s \geq 1$ and every $a \in [n]$ there is a **reward** $q_a^{(s)}$ (for choosing action a in round s), where $0 \leq q_a^{(s)} \leq 1$.
- ▶ We call $Q := (q_a^{(s)})_{a \in [n], s \geq 1}$ the **payoff matrix**. Usually, the number t of rounds is fixed in advance; then Q becomes an $n \times t$ matrix.

Formal Problem Description (cont'd)

- The maximal single-action reward after t rounds is

$$q_{\max}^{(t)} := \max_{a \in [n]} \sum_{s=1}^t q_a^{(s)}.$$

- The reward of a sequence of actions $\mathbf{a} = (a^{(1)}, \dots, a^{(t)}) \in [n]^t$ is

$$q(\mathbf{a}) := \sum_{s=1}^t q_{a^{(s)}}^{(s)}$$

and the regret of \mathbf{a} is

$$r(\mathbf{a}) := q_{\max}^{(t)} - q(\mathbf{a}).$$

Formal Problem Description (cont'd)

- ▶ An **algorithm** (or **strategy**) A picks in each round t an action $a^{(t)}$ only depending on the actions $a^{(s)}$ and rewards $q^{(s)} := q_{a^{(s)}}^{(s)}$ of the previous rounds $s = 1, \dots, t - 1$.
- ▶ The choice may be random, that is, $a^{(s)}$ is drawn randomly according to some probability distribution $\mathcal{D}^{(s)}$ on $[n]$.
- ▶ The expected reward for A at time t is

$$q(A) := E(q(a^{(1)}, \dots, a^{(t)})),$$

and the expected regret at time t is

$$r(A) := E(r(a^{(1)}, \dots, a^{(t)})).$$

Multiplicative Weights Update Algorithm

Algorithm EXP3

Parameter: γ , where $0 < \gamma \leq 1$.

Initialisation: $w_a^{(1)} := 1$ for all $a \in [n]$

1. for $s = 1, 2, \dots, t$ do
2. $\mathcal{D}^{(s)}$ probability distribution defined by

$$\mathcal{D}^{(s)}(\{a\}) := p_a^{(s)} := (1 - \gamma) \frac{w_a^{(s)}}{\sum_{a'=1}^n w_{a'}^{(s)}} + \frac{\gamma}{n}$$

3. action $a^{(s)}$ drawn randomly from $\mathcal{D}^{(s)}$
4. reward $q^{(s)} \leftarrow q_{a^{(s)}}^{(s)}$
5. weights are updated as follows:

$$w_a^{(s+1)} \leftarrow \begin{cases} w_a^{(s)} \cdot \exp\left(\frac{\gamma q^{(s)}}{np_a^{(s)}}\right) & \text{if } a = a^{(s)}, \\ w_a^{(s)} & \text{otherwise} \end{cases}$$

- ▶ The parameter γ determines the tradeoff between exploration and exploitation: the closer γ gets to 1, the more weight we put on exploration.
- ▶ The intuition behind the weight-update factor $\exp\left(\frac{\gamma q^{(s)}}{np_a^{(s)}}\right)$ is as follows:
 - ▶ if the reward $q^{(s)}$ is higher, the update factor is higher;
 - ▶ if an action is less likely ($p_a^{(s)}$ is smaller), the update factor is higher; this is to counteract that for these events the updates are less frequent (in expectation).

The precise definition of the factor falls out of the calculations in the proof of Theorem 4.7.

- ▶ The name Exp3 stands for **exponential-weight algorithm for exploration and exploitation**.

Theorem 4.7

For every payoff matrix, the expected regret of the Exp3 algorithm is bounded as follows:

$$r(\text{Exp3}) \leq (e - 1) \cdot \gamma \cdot q_{\max}^{(t)} + \frac{1}{\gamma} \cdot n \cdot \ln n.$$

(Proof omitted.)

Corollary 4.8

Set $\gamma^* := \min \left\{ 1, \sqrt{\frac{n \cdot \ln n}{(e - 1) \cdot q_{\max}^{(t)}}} \right\}$.

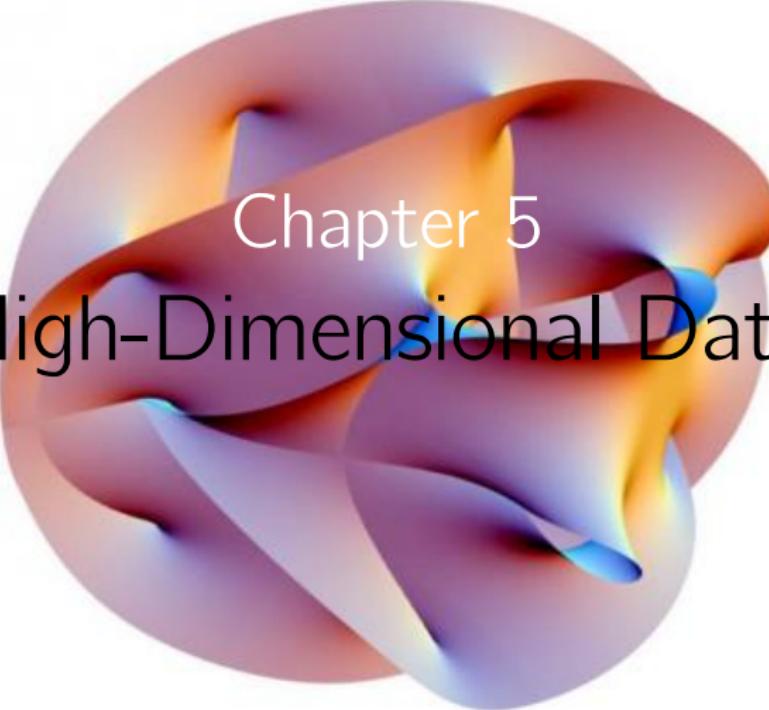
Then for every payoff matrix, the expected regret of the Exp3 algorithm with parameter γ^* satisfies

$$r(\text{Exp3}) \leq 2.63 \cdot \sqrt{q_{\max}^{(t)} \cdot n \cdot \ln n}.$$

Further Remarks

- ▶ The essential difference between the bandit learning scenario and the expert advice scenario we consider in the first section is that in the bandit learning scenario we cannot observe the reward (or loss) that the other actions would have given.
This explains the need for enforcing exploration, that is, adding the term γ/n to the weights in each round.
- ▶ The notion of regret we use here (comparing the reward of the algorithm with the reward of the best machine) is known as **weak regret**. There is also a notion of **strong regret**, which compares the reward of the algorithm with the reward of the best possible sequence of actions.
Obviously, bounds for strong regret will be weaker, and they are harder to obtain.
- ▶ We consider the bandit problem in an adversarial setting, that is, we bound the regret for the worst possible payoff matrix. There is also a stochastic version of the problem.

[Arora et al., 2012] is a survey on the multiplicative weights update algorithm and its applications. We follow this survey in our presentation of the algorithm. With a somewhat different notation, boosting is discussed in the survey as well; alternatively, it is also covered in [Shalev-Shwartz and Ben-David, 2014, Chapter 10] and [Blum et al., 2020, Section 6.7]. For details on bandit learning and the Exp3 algorithm, see the original article [Auer et al., 2002].



Chapter 5

High-Dimensional Data

Goals of this Chapter

This chapter has two main goals:

- ▶ on the technical side, it introduces some more advanced mathematical concepts from geometry and linear algebra, maybe most importantly a treatment of eigenvalues and eigenvectors from an algorithmic angle;
- ▶ on the applied side, the main theme is dimensionality reduction.

5.1 The Strange Geometry of High-Dimensional Spaces

The Volume is Near the Surface

Let $X \subseteq \mathbb{R}^\ell$. By $\text{vol}(X)$ we denote the **volume** of X .

Note

$\text{vol}(X)$ is not defined for all sets X (only for **measurable** X).

In the following, we always assume that X is a measurable set.

Lemma 5.1

Let $X \subseteq \mathbb{R}^\ell$. Let $c \in \mathbb{R}$ and $cX := \{c\mathbf{x} \mid \mathbf{x} \in X\}$. Then

$$\text{vol}(cX) = c^\ell \text{vol}(X).$$

Theorem 5.2

Let $X \subseteq \mathbb{R}^\ell$ such that $\text{vol}(X) > 0$ and $0 \leq \varepsilon \leq 1$. Then

$$\frac{\text{vol}((1 - \varepsilon)X)}{\text{vol}(X)} \leq e^{-\varepsilon\ell}.$$

Since $e^{-\varepsilon\ell}$ converges to 0 quickly as ℓ gets large, this means that the volume of high-dimensional objects is concentrated near the surface.

Proof Sketch for the Lemma.

Consider a cube Q with side length s first, say,

$$Q = \{\mathbf{x} \mid 0 \leq x_i \leq s \text{ for all } i\}.$$

Then $\text{vol}(Q) = s^\ell$ and $\text{vol}(cQ) = (cs)^\ell = c^\ell \text{vol}(Q)$.

Now fill X with infinitesimally small cubes, and the result follows. □

Proof of the Theorem.

$$\text{vol}((1 - \varepsilon)X) = (1 - \varepsilon)^\ell \text{vol}(X) \leq e^{-\varepsilon\ell} \text{vol}(X).$$



Why Should We Care?

Example (k -Nearest Neighbour in High-Dimensional Space)

Suppose we have a training set S of m points in \mathbb{R}^ℓ uniformly distributed over the unit ball

$$B^\ell := \{\mathbf{x} \in \mathbb{R}^\ell \mid \|\mathbf{x}\| \leq 1\}.$$

Now consider a query point \mathbf{x} . The probability that a single training point has distance at most $d < 1$ from \mathbf{x} is at most

$$\begin{aligned} \frac{\text{vol}(\{\mathbf{x}' \in B^\ell \mid \|\mathbf{x}' - \mathbf{x}\| \leq d\})}{\text{vol}(B^\ell)} &\leq \frac{\text{vol}(\{\mathbf{x}' \in \mathbb{R}^\ell \mid \|\mathbf{x}' - \mathbf{x}\| \leq d\})}{\text{vol}(B^\ell)} \\ &= \frac{\text{vol}(\{\mathbf{x}' \in \mathbb{R}^\ell \mid \|\mathbf{x}'\| \leq d\})}{\text{vol}(B^\ell)} \\ &= \frac{d^\ell \text{vol}(B^\ell)}{\text{vol}(B^\ell)} \\ &= d^\ell \end{aligned}$$

Thus the expected number of points from the training set S that have distance at most d from the query point \mathbf{x} is at most

$$md^\ell.$$

To find the k nearest neighbours of \mathbf{x} in S , we want this to be at least k .

Now assume $\ell = 100$, $m = 10^6$, and $k = 10$. Then

$$\begin{aligned} md^\ell \geq k &\iff d^{100} \geq 10^{-5} \\ &\iff d \geq 10^{-1/20} \approx 0.89. \end{aligned}$$

Thus the ball around the query point we must look at to find the k nearest neighbours has almost the same radius as the whole space, which defies the idea of looking at training points “near” the query point to find out the correct classification.

High-Dimensional Unit Balls

The ℓ -dimensional **unit ball** is the set

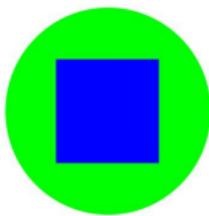
$$B^\ell := \{\mathbf{x} \in \mathbb{R}^\ell \mid \|\mathbf{x}\| \leq 1\}.$$



Theorem 5.3 (Volume of the Unit Ball)

$$\lim_{\ell \rightarrow \infty} \text{vol}(B^\ell) = 0.$$

Volume of the Unit Ball



$$\text{vol}(B^2) = \pi \approx 3.14$$

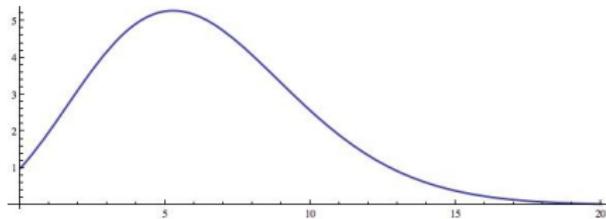


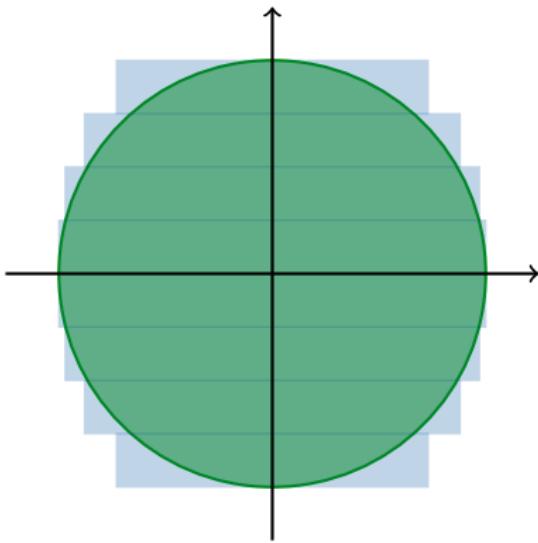
$$\text{vol}(B^3) = 4\pi/3 \approx 4.19$$

A table with the first few values:

ℓ	1	2	3	4	5	6	7	8
$\text{vol}(B^\ell)$	2	π	$\frac{4\pi}{3}$	$\frac{\pi^2}{2}$	$\frac{8\pi^2}{15}$	$\frac{\pi^3}{6}$	$\frac{16\pi^3}{105}$	$\frac{\pi^4}{24}$
Appr.	2	3.14	4.19	4.93	5.26	5.17	4.72	4.06

Plot of values $\ell = 1, \dots, 20$:





Proof sketch.

Cover B^ℓ by $2k$ cylinders.

Thickness of cylinder:

$$t := \frac{1}{k}$$

Radius of i th cylinder above (or below) equator:

$$r_i := \sqrt{1 - ((i-1)t)^2}.$$

Volume of i th cylinder:

$$t \cdot r_i^{\ell-1} \cdot \text{vol}(B^{\ell-1}).$$

Thus

$$\text{vol}(B^\ell) \leq 2 \sum_{i=1}^k t \cdot r_i^{\ell-1} \cdot \text{vol}(B^{\ell-1})$$

$$= \underbrace{\left(\frac{2}{k} \sum_{i=1}^k \left(1 - \left(\frac{i-1}{k} \right)^2 \right)^{\frac{\ell-1}{2}} \right)}_{=: f_k(\ell)} \cdot \text{vol}(B^{\ell-1})$$

To prove that $\lim_{\ell \rightarrow \infty} \text{vol}(B^\ell) = 0$, it suffices to prove that for some k there is a $c < 1$ such that $f_k(\ell) \leq c$ for all sufficiently large ℓ .

We choose $k := 4$ and $c := 3/4$. We have

$$\begin{aligned} f_4(\ell) &= \frac{1}{2} \sum_{i=1}^4 \left(1 - \left(\frac{i-1}{4} \right)^2 \right)^{\frac{\ell-1}{2}} \\ &= \frac{1}{2} + \frac{1}{2} \sum_{i=1}^3 \left(1 - \frac{i^2}{16} \right)^{\frac{\ell-1}{2}} \\ &\leq \frac{1}{2} + \frac{3}{2} \left(\frac{15}{16} \right)^{\frac{\ell-1}{2}}. \end{aligned}$$

For sufficiently large ℓ , we have $\left(\frac{15}{16} \right)^{\frac{\ell-1}{2}} \leq \frac{1}{6}$ and thus $\frac{3}{2} \left(\frac{15}{16} \right)^{\frac{\ell-1}{2}} \leq \frac{1}{4}$, which implies $f_4(\ell) \leq 3/4$. □

Concentration Near Equator

Theorem 5.4

Let $\ell \geq 3$ and $c \geq 1$. Then

$$\frac{\text{vol} \left(\left\{ \mathbf{x} \in B^\ell \mid |x_1| > \frac{c}{\sqrt{\ell-1}} \right\} \right)}{\text{vol}(B^\ell)} \leq \frac{2}{c} e^{-c^2/2}.$$

That is, at least a $(1 - \frac{2}{c} e^{-c^2/2})$ -fraction of the volume of the unit ball B^ℓ has distance at most $\frac{c}{\sqrt{\ell-1}}$ from the “equator” half-plane $x_1 = 0$.

Corollary 5.5

Let $\ell \geq 3$ and $c \geq 1$, and let $\mathbf{a} \in \mathbb{R}^\ell$ be an arbitrary unit vector (that is, $\|\mathbf{a}\| = 1$). Then

$$\frac{\text{vol} \left(\left\{ \mathbf{x} \in B^\ell \mid |\langle \mathbf{a}, \mathbf{x} \rangle| > \frac{c}{\sqrt{\ell-1}} \right\} \right)}{\text{vol}(B^\ell)} \leq \frac{2}{c} e^{-c^2/2}.$$

A Probabilistic View on the Equator

- ▶ Concentration near the equator, no matter which point we choose as the north pole, is a very strange result if thought of geometrically.
- ▶ If we think about it probabilistically, it is fairly obvious.
- ▶ Think of drawing a point $\mathbf{x} = (x_1, \dots, x_\ell) \in B^\ell$ uniformly at random.
- ▶ On average, $|x_i|$ will be around $1/\sqrt{\ell}$ or less, because otherwise the length $\|\mathbf{x}\|$ is too large.
- ▶ Thus the probability of $|x_1|$ being significantly higher than $1/\sqrt{\ell}$ is low.

A Probabilistic View on the Volume

- ▶ What is the probability p_ℓ that a vector \mathbf{x} uniformly chosen from the cube

$$Q^\ell := \{\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{R}^\ell \mid |x_i| \leq 1 \text{ for all } i \in [\ell]\}$$

is in B^ℓ , that is, has length $\|\mathbf{x}\| \leq 1$?

- ▶ We have

$$\text{vol}(B^\ell) = \text{vol}(Q^\ell) \cdot p_\ell = 2^\ell p_\ell$$

Hence $\lim_{\ell \rightarrow \infty} \text{vol}(B^\ell) = 0$ is equivalent to $p_\ell = o(2^{-\ell})$.

- ▶ To choose a vector $\mathbf{x} = (x_1, \dots, x_\ell)$ uniformly from Q^ℓ , we can choose the x_i independently from the closed interval $[-1, 1]$.
- ▶ Thus indeed, p_ℓ is very low, because to have $\|\mathbf{x}\| \leq 1$, most coordinates x_i must have absolute value around $1/\sqrt{\ell}$ or below. However, the expected value for $|x_i|$ is $1/2$.
- ▶ We may also view this as a tail bound: the expected value for $\|\mathbf{x}\|^2$ is $\ell/3$, and we want to prove $p_\ell = \Pr(\|\mathbf{x}\|^2 \leq 1) = o(2^{-\ell})$.

Expected value of $\|\mathbf{x}\|^2$:

$$\mathsf{E}(\|\mathbf{x}\|^2) = \mathsf{E}\left(\sum_{i=1}^{\ell} x_i^2\right) = \sum_{i=1}^{\ell} \mathsf{E}(x_i^2) = \ell \int_{-1}^1 \frac{1}{2} x^2 dx = \frac{\ell}{3}.$$

5.2 Dimension Reduction by Random Projections

Suppose we have a set $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ of points in high-dimensional Euclidean space \mathbb{R}^ℓ .

- ▶ In this section, we'll see a technique for mapping \mathbb{R}^ℓ to a low-dimensional space \mathbb{R}^k , where $k \approx \log n$, while approximately preserving the distances between all pairs of points in X .
- ▶ We can even do this by a linear mapping. In fact, we shall prove that a randomly chosen linear mapping will work with high probability. This is the content of the **Johnson-Lindenstrauss Lemma**.
- ▶ More precisely, we specify the mapping by choosing the entries of a $(k \times \ell)$ -matrix independently according to a normal distribution. Equivalently, we can think of this as choosing the rows of the matrix independently according to an ℓ -dimensional spherical Gaussian distribution.

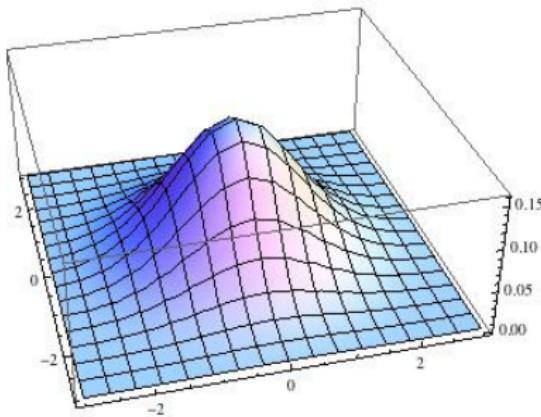
Spherical Gaussian Distribution

An ℓ -dimensional **spherical Gaussian distribution** with mean $\mu \in \mathbb{R}^\ell$ and variance σ^2 in each direction is the probability distribution on \mathbb{R}^ℓ with density

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

Example 5.6

2-dimension spherical Gaussian with mean $\mu = \mathbf{0}$ and variance $\sigma^2 = 1$:



Spherical Gaussian Distribution (cont'd)

Lemma 5.7

A spherical Gaussian distribution is obtained by choosing the coordinates independently from a 1-dimensional normal distribution.

More precisely, the ℓ -dimensional spherical Gaussian distribution with mean $\mu = (\mu_1, \dots, \mu_\ell) \in \mathbb{R}^\ell$ and variance σ^2 in each direction is the same distribution as the one obtained by drawing the coordinates x_i of $\mathbf{x} = (x_1, \dots, x_\ell)$ independently according to a normal distribution with mean μ_i and variance σ^2 .

Remark 5.8

The spherical Gaussian is a special case of the **multivariate normal distribution** where the coordinates are independent and have the same variance, which implies that the contours are spheres.

Gaussian Annulus Theorem

In a 1-dimensional Gaussian distribution most of the probability mass is near the mean.

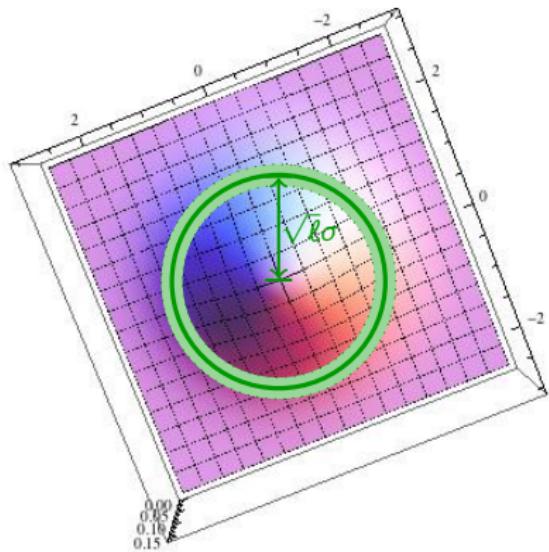
This is different for high-dimensional spherical Gaussians, where the probability mass is concentrated in a thin annulus of radius $\sigma\sqrt{\ell}$ around the mean.

Theorem 5.9

Let $b \leq \sqrt{\ell}$, and let $\mathbf{x} \in \mathbb{R}^\ell$ be drawn from an ℓ -dimensional spherical Gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. Then

$$\Pr(\sqrt{\ell}-b < \|\mathbf{x}\| < \sqrt{\ell}+b) \geq 1-3e^{-cb^2},$$

for a constant $c > 0$ not depending on ℓ and b .



Proof idea.

For $\mathbf{x} = (x_1, \dots, x_\ell)$, distributed according to the ℓ -dimensional Gaussian distribution with mean $\mathbf{0}$ and variance σ^2 in each direction, we have

$$\begin{aligned}\mathbb{E}(\|\mathbf{x}\|^2) &= \mathbb{E}\left(\sum_{i=1}^{\ell} x_i^2\right) = \sum_{i=1}^{\ell} \mathbb{E}(x_i^2) \\ &= \ell\sigma^2 && \text{because } \mathbb{E}(x_i^2) \text{ is the variance of } x_i \\ &= \ell.\end{aligned}$$

Thus, as for the volume of the unit ball, we need to prove a tail bound. We use the fact that in a spherical Gaussian the coordinates are independent and apply Theorem 2.8. □

The Reduction Mapping

In the following, we let $k, \ell \in \mathbb{R}$, where $k \leq \ell$.

We draw vectors $\mathbf{u}_1, \dots, \mathbf{u}_k \in \mathbb{R}^\ell$ independently from the ℓ -dimensional spherical Gaussian distribution with mean $\mathbf{0}$ and variance 1 in each direction and let

$$U := \frac{1}{\sqrt{k}} \begin{pmatrix} \mathbf{u}_1^\top \\ \vdots \\ \mathbf{u}_k^\top \end{pmatrix} \in \mathbb{R}^{k \times \ell}.$$

The mapping $\mathbf{x} \mapsto U\mathbf{x}$ is the “random projection” that we use for dimension reduction.

The Random Projection Theorem

Theorem 5.10

For all $\mathbf{x} \in \mathbb{R}^\ell$ and all $\varepsilon > 0$,

$$\Pr \left(\left| \|U\mathbf{x}\| - \|\mathbf{x}\| \right| > \varepsilon \|\mathbf{x}\| \right) \leq 3e^{-c\varepsilon^2 k},$$

where the probability is over the choice of the vectors $\mathbf{u}_1, \dots, \mathbf{u}_k$ used to construct the matrix U and c is the constant from the Gaussian Annulus Theorem.

Corollary 5.11

For all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^\ell$ and all $\varepsilon \in (0, 1)$,

$$\Pr \left((1 - \varepsilon) \|\mathbf{x} - \mathbf{y}\| \leq \|U\mathbf{x} - U\mathbf{y}\| \leq (1 + \varepsilon) \|\mathbf{x} - \mathbf{y}\| \right) \geq 1 - 3e^{-c\varepsilon^2 k}.$$

Combining Normally Distributed Random Variables

Without proof we use the following fact about normal distributions.

Lemma 5.12

Let X_1, \dots, X_ℓ be independent random variables that are normally distributed with mean 0 and variance 1. Furthermore, let $\mathbf{a} = (a_1, \dots, a_\ell) \in \mathbb{R}^\ell$ and $X = \sum_{i=1}^{\ell} a_i X_i$. Then X is normally distributed with mean 0 and variance $\|\mathbf{a}\|^2$.

Proof of the Random Projection Theorem.

Note that the assertion is trivial for $\mathbf{x} = \mathbf{0}$. Hence without loss of generality we assume $\mathbf{x} \neq \mathbf{0}$. Then we may assume that $\|\mathbf{x}\| = 1$, because we can divide both sides of the inequality by $\|\mathbf{x}\|$ without changing the probability.

Suppose that $\mathbf{u}_i = (u_{i1}, \dots, u_{i\ell})^\top$ and $\mathbf{x} = (x_1, \dots, x_\ell)^\top$. The u_{ij} are independent random variables normally distributed with mean 0 and variance σ^2 . Hence by Lemma 5.12,

$$\langle \mathbf{u}_i, \mathbf{x} \rangle = \sum_{j=1}^{\ell} u_{ij} x_j$$

is normally distributed with mean 0 and variance $\|\mathbf{x}\|^2 = 1$.

Thus the vector

$$\mathbf{y} := \begin{pmatrix} \langle \mathbf{u}_1, \mathbf{x} \rangle \\ \vdots \\ \langle \mathbf{u}_k, \mathbf{x} \rangle \end{pmatrix}$$

is distributed according to a k -dimensional spherical Gaussian with mean $\mathbf{0}$ and variance 1 in each direction. Observe that $\|U\mathbf{x}\| = \frac{1}{\sqrt{k}}\|\mathbf{y}\|$.

By the Gaussian Annulus Theorem with $\ell:=k$, $b:=\varepsilon\sqrt{k}$,

$$\begin{aligned} & \Pr(\sqrt{k} - \varepsilon\sqrt{k} \leq \|\mathbf{y}\| \leq \sqrt{k} + \varepsilon\sqrt{k}) \geq 1 - 3e^{-c\varepsilon^2 k} \\ \iff & \Pr(|\|\mathbf{y}\| - \sqrt{k}| > \varepsilon\sqrt{k}) < 3e^{-c\varepsilon^2 k} \\ \iff & \Pr(|\|U\mathbf{x}\| - 1| > \varepsilon) < 3e^{-c\varepsilon^2 k}. \end{aligned}$$

Since $\|\mathbf{x}\| = 1$, this completes the proof. □

The Johnson-Lindenstrauss Lemma

Corollary 5.13 (Johnson-Lindenstrauss Lemma)

Let $0 < \varepsilon < 1$ and $k, \ell, n \in \mathbb{N}$ such that $k \geq \frac{3}{c\varepsilon^2} \ln n$, where c is the constant from the Gaussian Annulus Theorem.

Then for every set $X \subseteq \mathbb{R}^\ell$ of size $|X| = n$,

$$\Pr \left(\forall \mathbf{x}, \mathbf{y} \in X : (1 - \varepsilon) \|\mathbf{x} - \mathbf{y}\| \leq \|U\mathbf{x} - U\mathbf{y}\| \leq (1 + \varepsilon) \|\mathbf{x} - \mathbf{y}\| \right) \geq 1 - \frac{3}{2n}.$$

Proof of the Johnson-Lindenstrauss Lemma.

We apply the Union Bound to Corollary 5.11:

$$\begin{aligned} & \Pr \left(\exists \mathbf{x}, \mathbf{y} \in X : (1 - \varepsilon) \|\mathbf{x} - \mathbf{y}\| > \|U\mathbf{x} - U\mathbf{y}\| \text{ or } \|U\mathbf{x} - U\mathbf{y}\| > (1 + \varepsilon) \|\mathbf{x} - \mathbf{y}\| \right) \\ & \leq \binom{n}{2} 3e^{-c\varepsilon^2 k} \leq \frac{3n^2}{2} e^{-3\ln n} = \frac{3}{2n}. \end{aligned}$$

□

5.3 Background from Linear Algebra: Eigenvalues and Eigenvectors

Eigenvalues and Eigenvectors

Let $A \in \mathbb{C}^{n \times n}$ be a square matrix.

An eigenpair of A is a pair $(\lambda, \mathbf{u}) \in \mathbb{C} \times (\mathbb{C}^n \setminus \{\mathbf{0}\})$ such that

$$A\mathbf{u} = \lambda\mathbf{u}.$$

λ is called an eigenvalue of A and \mathbf{u} an eigenvector associated with λ .

The eigenvectors associated with an eigenvalue λ together with $\mathbf{0}$ form a linear subspace of \mathbb{C}^n called the eigenspace of λ . The geometric multiplicity of λ is the dimension of its eigenspace.

Remark 5.14

We are mainly interested in real eigenvalues of real matrices here, but in general the theory is better developed over the complex numbers.

Eigenvalues and the Characteristic Polynomial

Fact 5.15

The eigenvalues of $A \in \mathbb{C}^{n \times n}$ are precisely the zeroes of the **characteristic polynomial** $\det(A - xl)$, where I is the $n \times n$ identity matrix and x an indeterminate.

Remark 5.16

This fact gives us a way of computing the eigenvalues of a matrix. However, this method is numerically unstable and too inefficient for large matrices.

The Spectrum

The characteristic polynomial $\det(A - xI)$ of a matrix $A \in \mathbb{C}^{n \times n}$ is a polynomial of degree n that factors as

$$\det(A - xI) = \prod_{i=1}^n (\lambda_i - x). \quad (*)$$

over \mathbb{C} . The λ_i are the zeroes of the polynomial (and thus the eigenvalues of A).

The set $\{\lambda_1, \dots, \lambda_n\}$ is called the **spectrum** of A . The number of times the factor $(\lambda_i - x)$ appears in the product $(*)$ is called the **algebraic multiplicity** of λ_i .

The **spectral radius** $\rho(A)$ is the maximum absolute value of an eigenvalue of A .

Remark 5.17

It is easy to see that the algebraic multiplicity of an eigenvalue is always greater than or equal to the geometric multiplicity; the converse does not necessarily hold.

Diagonisable Matrices

By $\text{diag}(d_1, \dots, d_n)$ we denote the $(n \times n)$ -diagonal matrix with diagonal entries d_1, \dots, d_n .

A matrix $A \in \mathbb{C}^{n \times n}$ is **diagonalisable** if there are a nonsingular matrix U and a diagonal matrix Λ such that $U^{-1}AU = \Lambda$.

Theorem 5.18

$A \in \mathbb{C}^{n \times n}$ is diagonalisable if and only if \mathbb{C}^n has a basis consisting of eigenvectors of A .

More precisely:

- (1) If $U^{-1}AU = \Lambda$, where $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$, then $\lambda_1, \dots, \lambda_n$ is the spectrum of A , and if the columns of U are $\mathbf{u}_1, \dots, \mathbf{u}_n$, then \mathbf{u}_i is an eigenvector of A associated with λ_i .
- (2) If $\lambda_1, \dots, \lambda_n$ are the eigenvalues of A , $\mathbf{u}_1, \dots, \mathbf{u}_n$ is a basis of corresponding eigenvectors, $U \in \mathbb{C}^{n \times n}$ is the matrix with columns $\mathbf{u}_1, \dots, \mathbf{u}_n$, and $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$, then $U^{-1}AU = \Lambda$.

Proof.

To prove (1), suppose that $U^{-1}AU = \Lambda$ for a diagonal matrix Λ . Then

$$AU = U\Lambda.$$

Now suppose that $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$, and let $\mathbf{u}_1, \dots, \mathbf{u}_n$ be the columns of U . Then

$$A\mathbf{u}_i = (AU)_i = (U\Lambda)_i = \lambda_i \mathbf{u}_i.$$

Here $(AU)_i$ and $(U\Lambda)_i$ denote the i th columns of the respective matrices.

Thus $(\lambda_i, \mathbf{u}_i)$ is an eigenpair. Moreover, since U is invertible the \mathbf{u}_i are linearly independent and thus form a basis of \mathbb{C}^n .

The proof of (2) is similar. We have $(AU)_i = A\mathbf{u}_i = \lambda_i \mathbf{u}_i = (U\Lambda)_i$ and thus $AU = U\Lambda$. Since the \mathbf{u}_i form a basis of \mathbb{C}^n , the matrix U is invertible, and thus $U^{-1}AU = \Lambda$. □

Spectral Decomposition

Recall that an **orthonormal basis** of \mathbb{R}^n is a set $\mathbf{u}_1, \dots, \mathbf{u}_n \in \mathbb{R}^n$ such that $\|\mathbf{u}_i\| = 1$ for all i and $\langle \mathbf{u}_i, \mathbf{u}_j \rangle = 0$ for all $i \neq j$.

Theorem 5.19

Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then all eigenvalues of A are real, and \mathbb{R}^n has an orthonormal basis consisting of eigenvectors of A .

(Proof omitted.)

A matrix $U \in \mathbb{R}^{n \times n}$ is **orthogonal** if $U^{-1} = U^\top$, or equivalently, if the columns of U form an orthonormal basis of \mathbb{R}^n .

Corollary 5.20 (Spectral Decomposition)

Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then there is an orthogonal matrix $U \in \mathbb{R}^n$ such that

$$A = U \Lambda U^\top,$$

where Λ is the diagonal matrix whose diagonal entries form the spectrum of A .

Perron-Frobenius Theorem

With each matrix $A = (A_{ij}) \in \mathbb{R}^{n \times n}$ we associate a directed graph G_A with vertex set $V(G_A) := [n]$ and edge set

$$E(G_A) := \{(i, j) \mid A_{ij} \neq 0\}.$$

A is **irreducible** if G_A is strongly connected.

Theorem 5.21 (Perron-Frobenius)

Let $n \geq 2$, and let $A \in \mathbb{R}^{n \times n}$ be nonnegative and irreducible with spectral radius $\rho := \rho(A)$. Then

- (1) ρ is an eigenvalue of A of algebraic multiplicity 1.
- (2) There is a unique eigenvector $\mathbf{u} \in \mathbb{R}^n$ associated with ρ such that $\|\mathbf{u}\| = 1$ and all entries of \mathbf{u} are positive.
- (3) There is a unique vector $\mathbf{v} \in \mathbb{R}^n$ such that $\mathbf{v}^\top A = \rho \mathbf{v}^\top$ and $\|\mathbf{v}\| = 1$ and all entries of \mathbf{v} are positive.

The vectors \mathbf{u} and \mathbf{v} are called the right and left **Perron vectors** of A .
(Proof omitted.)

Let $A \in \mathbb{R}^{n \times n}$. For a permutation π of $[n]$, we let A^π be the matrix with entries $A_{ij}^\pi := A_{\pi^{-1}(i)\pi^{-1}(j)}$. This is the matrix obtained from A by simultaneously permuting rows and columns with π .

Let us call A **reducible** if there are a $k \in [n - 1]$ and matrices $B \in \mathbb{R}^{k \times k}$, $C \in \mathbb{R}^{k \times (n-k)}$ and $D \in \mathbb{R}^{(n-k) \times (n-k)}$, and a permutation π of $[n]$ such that

$$A^\pi = \begin{pmatrix} B & C \\ 0 & D \end{pmatrix}.$$

Lemma 5.22

For every nonnegative matrix $A \in \mathbb{R}^{n \times n}$ the following are equivalent.

- (i) A is irreducible.
- (ii) A is not reducible.
- (iii) $(A + I)^{n-1}$ is positive, that is, has only positive entries.

(Proof as an exercise.)

Limit Theorem for Nonnegative Matrices

Theorem 5.23

Let $n \geq 2$, and let $A \in \mathbb{R}^{n \times n}$ be nonnegative and irreducible with spectral radius $\rho := \rho(A)$. Let \mathbf{u}, \mathbf{v} be the right and left Perron vectors of A . Then

$$\lim_{k \rightarrow \infty} \frac{1}{k} \sum_{i=1}^k \frac{A^i}{\rho^i} = \frac{1}{\langle \mathbf{u}, \mathbf{v} \rangle} \mathbf{u} \cdot \mathbf{v}^\top$$

(Proof omitted.)

5.4 Power Iteration

Assumptions

Let $A \in \mathbb{C}^{n \times n}$ be a matrix with spectrum $\lambda_1, \dots, \lambda_n$, where

$$|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|.$$

Assumption 5.24

$\lambda_1 \in \mathbb{R}_{\geq 0}$ and $\lambda_1 > |\lambda_2|$.

This implies that $\rho(A) = \lambda_1$ and that the algebraic multiplicity of λ_1 is 1.

Assumption 5.25

A is diagonalisable.

Let $\mathbf{u}_1, \dots, \mathbf{u}_n$ be a basis of \mathbb{C}^n such that $\|\mathbf{u}_i\| = 1$ and \mathbf{u}_i is an eigenvector of A associated with λ_i .

Let $\mathbf{x} \in \mathbb{C}^n$ and for $k \geq 0$,

$$\mathbf{x}_k := A^k \mathbf{x}.$$

For all $k \in \mathbb{N}$, let $\hat{\mathbf{x}}_k := \frac{\mathbf{x}_k}{\|\mathbf{x}_k\|}$.

Theorem 5.26

Assume \mathbf{x} is not orthogonal to \mathbf{u}_1 . Then for $k \rightarrow \infty$ the sequence $(\hat{\mathbf{x}}_k)$ converges to an eigenvector of A associated with λ_1 .

Proof.

We write \mathbf{x} in the basis $\mathbf{u}_1, \dots, \mathbf{u}_n$:

$$\mathbf{x} = \sum_{i=1}^n a_i \mathbf{u}_i,$$

for some $a_1, \dots, a_n \in \mathbb{C}^n$. Then $a_i = \langle \mathbf{x}, \mathbf{u}_i \rangle$. Thus $a_1 \neq 0$, because \mathbf{x} is not orthogonal to \mathbf{u}_1 .

We have

$$\begin{aligned}\mathbf{x}_k &= \sum_{i=1}^n a_i A^k \mathbf{u}_i \\ &= \sum_{i=1}^n a_i \lambda_i^k \mathbf{u}_i \\ &= a_1 \lambda_1^k \left(\mathbf{u}_1 + \sum_{i=2}^n \left(\frac{a_i}{a_1} \right) \left(\frac{\lambda_i}{\lambda_1} \right)^k \mathbf{u}_i \right).\end{aligned}$$

Since $0 \leq |\lambda_i/\lambda_1| < 1$ for all $i \geq 2$, we have

$$\sum_{i=2}^n \left(\frac{a_i}{a_1} \right) \left(\frac{\lambda_i}{\lambda_1} \right)^k \mathbf{u}_i \xrightarrow[k \rightarrow \infty]{} \mathbf{0}.$$

Thus

$$\frac{1}{a_1 \lambda_1^k} \mathbf{x}_k \xrightarrow[k \rightarrow \infty]{} \mathbf{u}_1.$$

Since $\|\mathbf{u}_1\| = 1$ and $\lambda_1 = |\lambda_1| \in \mathbb{R}_{\geq 0}$, this implies

$$\frac{1}{|a_1| \lambda_1^k} \|\mathbf{x}_k\| \xrightarrow[k \rightarrow \infty]{} 1$$

and hence

$$\frac{|a_1|}{a_1} \widehat{\mathbf{x}}_k = \frac{|a_1| \cdot \lambda_1^k \cdot \mathbf{x}_k}{a_1 \cdot \lambda_1^k \cdot \|\mathbf{x}_k\|} \xrightarrow[k \rightarrow \infty]{} \mathbf{u}_1.$$

Thus $(\widehat{\mathbf{x}}_k)$ converges to $\frac{a_1}{|a_1|} \mathbf{u}_1$, which is an eigenvector of A associated with λ_1 . □

The Power Iteration Algorithm

Algorithm POWER-ITERATION

Input: Matrix $A \in \mathbb{C}^{n \times n}$, vector $\mathbf{x} \in \mathbb{C}^n$

Output: Vector $\mathbf{v} \in \mathbb{C}^n$

1. $\mathbf{v}_0 \leftarrow \mathbf{x}/\|\mathbf{x}\|$
2. $k \leftarrow 0$
3. repeat
4. $k \leftarrow k + 1$
5. $\mathbf{v}_k \leftarrow \frac{A\mathbf{v}_{k-1}}{\|A\mathbf{v}_{k-1}\|}$
6. until the sequence converges within some tolerance
7. return \mathbf{v}_k

The theorem shows that the power iteration method converges to some eigenvector, provided that the matrix A satisfies Assumptions 5.24 and 5.25 and the initial vector \mathbf{x} is not orthogonal to \mathbf{u}_1 .

But it leaves two important questions open.

- ▶ How can we guarantee that \mathbf{x} is not orthogonal to \mathbf{u}_1 ?

If we choose $\mathbf{x} = \sum_{i=1}^n a_i \mathbf{u}_i$ randomly, then with high probability a_1 will be nonzero. However, the quality of our solution will also depend on a_1 not being too small. We need to choose \mathbf{x} randomly with a suitable distribution to guarantee this.

- ▶ When do we stop, and how fast does the sequence $(\mathbf{v}_k)_{k \geq 0}$ converge?

Our proof shows that the rate of convergence is determined by $\frac{|\lambda_2|}{\lambda_1} = \min_{i \geq 2} \frac{|\lambda_i|}{\lambda_1}$.

5.5 Principal Component Analysis

Focusing on the Important Features

Suppose we have a sequence of n data points $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbb{R}^\ell$, where the ℓ coordinates represent the **features** of our data points. We usually represent the points by the **data matrix** $A \in \mathbb{R}^{n \times \ell}$ whose rows are $\mathbf{a}_1^\top, \dots, \mathbf{a}_n^\top$.

Idea

To reduce the dimension, we focus on the most important features.

Feature selection: Select $k \leq \ell$ of the features and consider the $(n \times k)$ -submatrix corresponding to these features.

Feature extraction: Combine the ℓ given features into k new ones and create an $(n \times k)$ -matrix using these new features.
The most important way to combine features is by linear combinations.

Feature Selection and Extraction

Both feature selection and feature extraction reduce the original ℓ -dimensional data set to a new k -dimensional data set, for some $k \leq \ell$.

Clearly, we want the new data set to approximate the old one in a best possible way.

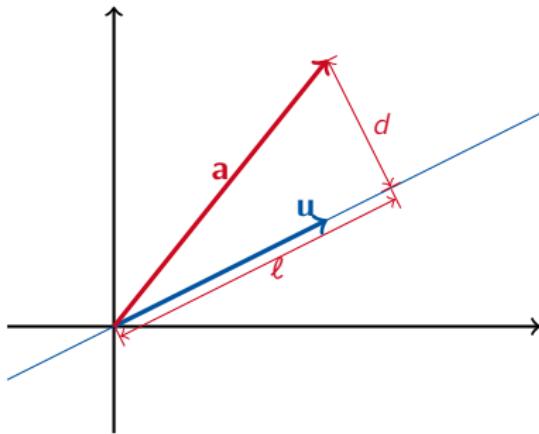
But what does this mean?

- ▶ We can try to **minimise the error**, that is, minimise the distance between the original and the new data set with respect to some metric.
- ▶ We can try to **maximise the variance** of the data, that is, extract features that highlight the differences between the data items.

It turns out that these two strategies are essentially the same.

Observation

Minimising squared distance = maximising squared lengths of projections



The longer the projection of \mathbf{a} to the line through \mathbf{u} the shorter the distance d . Recall that if $\|\mathbf{u}\| = 1$ then the length ℓ of the projection is $\langle \mathbf{a}, \mathbf{u} \rangle$.

Centering the Data

It is easiest to first normalise the data $\mathbf{a}_1, \dots, \mathbf{a}_n$ (the rows of the data matrix A) in such a way that the **mean** (a.k.a. **centroid**)

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{a}_i$$

is zero. We call a data sequence $\mathbf{a}_1, \dots, \mathbf{a}_n$ with mean **0**, or the corresponding data matrix, **centred**.

We can achieve this by simply replacing each data point \mathbf{a}_i by

$$\mathbf{a}'_i := \mathbf{a}_i - \boldsymbol{\mu}.$$

Note that $\boldsymbol{\mu}' = \frac{1}{n} \sum_{i=1}^n \mathbf{a}'_i = 0$. Thus $\mathbf{a}'_1, \dots, \mathbf{a}'_n$ is centred.

In the following, we usually assume that our data matrix A is centred.

The **variance** of a data sequence is the sum of the squared distances from the mean.

Thus if the data sequence $\mathbf{a}_1, \dots, \mathbf{a}_n$ is centred, its **variance** is

$$\sum_i \|\mathbf{a}_i\|^2.$$

Observation 5.27

If $A \in \mathbb{R}^{n \times \ell}$ is our data matrix with rows $\mathbf{a}_1^\top, \dots, \mathbf{a}_n^\top$, and if $\mathbf{a}^1, \dots, \mathbf{a}^\ell$ are the columns of A , then

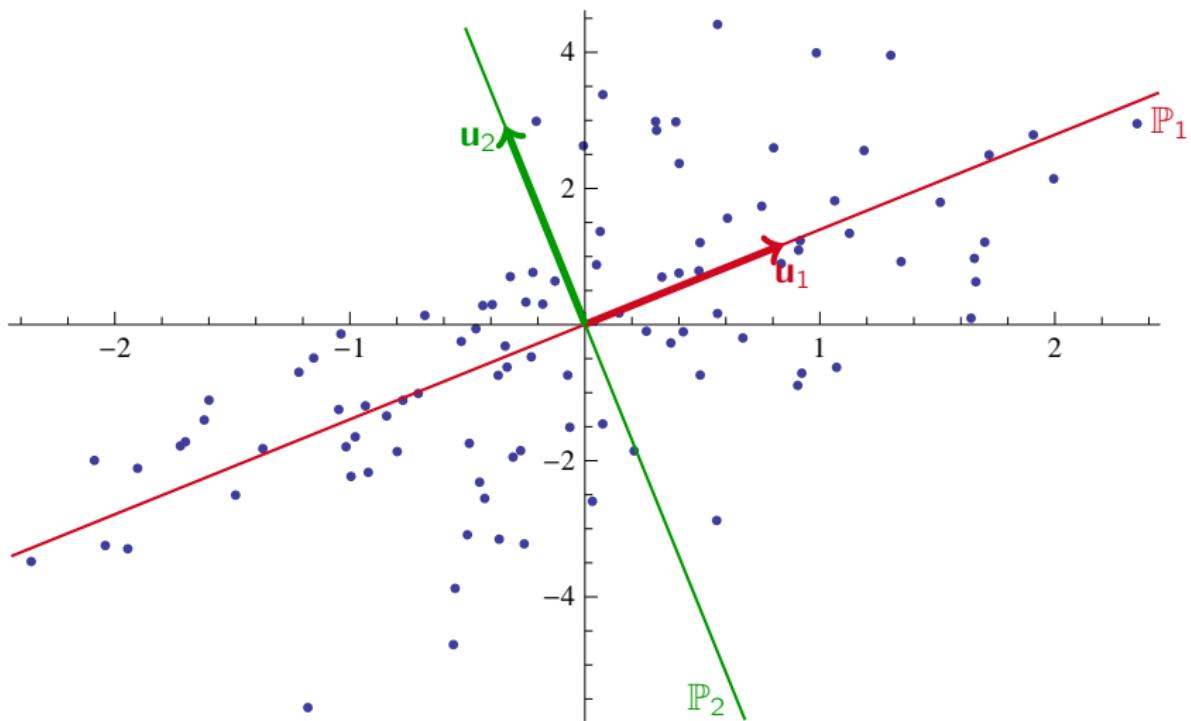
$$\sum_{i=1}^n \|\mathbf{a}_i\|^2 = \sum_{i=1}^n \sum_{j=1}^{\ell} A_{ij}^2 = \sum_{j=1}^{\ell} \|\mathbf{a}^j\|^2.$$

Principal Component Analysis

Let $A \in \mathbb{R}^{n \times \ell}$ be a matrix representing n data points in \mathbb{R}^ℓ .

- ▶ In **principal component analysis**, we apply an orthogonal transformation to our data, that is, we represent the data in a new orthonormal basis $\mathbf{u}_1, \dots, \mathbf{u}_\ell$.
- ▶ The vectors \mathbf{u}_i , or more precisely the lines $\mathbb{P}_i := \text{span}(\mathbf{u}_i)$ through these vectors, are the **principal components** of A .
- ▶ They are chosen as follows:
 - ▶ The first principal component \mathbb{P}_1 is the line in \mathbb{R}^ℓ that maximises the variance of the data, that is, if we project the data on \mathbb{P}_1 then the sum of the squares of the projection lengths is maximum.
 - ▶ After we have chosen the first i principal components $\mathbb{P}_1, \dots, \mathbb{P}_i$, the $(i + 1)$ -st principal component \mathbb{P}_{i+1} is the line in \mathbb{R}^ℓ that is orthogonal to $\mathbb{P}_1, \dots, \mathbb{P}_i$ and, subject to this condition, maximises the variance of the projected data.

Example



PCA Formally

Let $A \in \mathbb{R}^{n \times \ell}$ with rows $\mathbf{a}_1^\top, \dots, \mathbf{a}_n^\top$. A **PCA-transformation** of A is an orthogonal matrix $U \in \mathbb{R}^{\ell \times \ell}$ with columns $\mathbf{u}_1, \dots, \mathbf{u}_\ell$ satisfying the following conditions for all $j \in [\ell]$:

$$\mathbf{u}_j = \arg \max_{\substack{x \in \mathbb{R}^\ell \\ \text{such that } \|x\|=1 \\ \text{and } x \perp u_1, \dots, u_{j-1}}} \sum_{i=1}^n \langle \mathbf{a}_i, x \rangle^2.$$

The lines $\mathbb{P}_i = \text{span}(\mathbf{u}_i)$ are called the **principal components** of A w.r.t. the PCA transformation U .

Remark 5.28

The principal components of A are not unique. However, with the noisy data one usually finds in practice they tend to be.

If the principal components \mathbb{P}_i are unique, the vectors \mathbf{u}_i are unique up to their signs.

Notes for Page 5.39

$\sum_{i=1}^n \langle \mathbf{a}_i, \mathbf{x} \rangle^2$ is the variance of the data projected to $\text{span}(\mathbf{x})$. Note that

$$\sum_{i=1}^n \langle \mathbf{a}_i, \mathbf{x} \rangle^2 = \|A\mathbf{x}\|^2.$$

Best-Fit Subspaces

As usually, let $A \in \mathbb{R}^{n \times \ell}$ be our data matrix with rows $\mathbf{a}_1^\top, \dots, \mathbf{a}_n^\top$. We assume A to be centred. Furthermore, we let $1 \leq k \leq \ell$.

- ▶ A **best-fit k -dimensional subspace** for our data is a k -dimensional linear subspace $\mathbb{X} \subseteq \mathbb{R}^\ell$ such that the projection of $\mathbf{a}_1, \dots, \mathbf{a}_n$ into \mathbb{X} has maximum variance.
- ▶ If $\mathbf{x}_1, \dots, \mathbf{x}_k \in \mathbb{R}^\ell$ is an orthonormal basis of \mathbb{X} and $X \in \mathbb{R}^{\ell \times k}$ is the matrix with columns $\mathbf{x}_1, \dots, \mathbf{x}_k$, the rows of the matrix $AX \in \mathbb{R}^{n \times k}$ are precisely the projections of the data points into \mathbb{X} , represented in the basis $\mathbf{x}_1, \dots, \mathbf{x}_k$.
- ▶ Thus a best-fit k -dimensional subspace is a subspace $\mathbb{X} = \text{span}(\mathbf{x}_1, \dots, \mathbf{x}_k)$, where $\mathbf{x}_1, \dots, \mathbf{x}_k$ is an orthonormal system maximising

$$\sum_{i=1}^n \sum_{j=1}^k \langle \mathbf{a}_i, \mathbf{x}_j \rangle^2.$$

- ▶ We can use best-fit subspaces for dimension reduction: the projection of our data into a best-fit k -dimensional subspace is, in some sense, an optimal k -dimensional approximation of our original data set.

Dimension Reduction via PCA

Observe that the first principal component \mathbb{P}_1 of A (with respect to some PCA-transformation) is a best-fit 1-dimensional subspace for A .

Theorem 5.29

Let $A \in \mathbb{R}^{n \times \ell}$, and let U be a PCA-transformation of A with columns $\mathbf{u}_1, \dots, \mathbf{u}_\ell$. Then for every $k \in [\ell]$,

$$\mathbb{U}_k := \text{span}(\mathbf{u}_1, \dots, \mathbf{u}_k)$$

is a best-fit k -dimensional subspace for A .

Proof.

The proof is by induction on k .

The base case $k = 1$ is immediate from the definition of PCA-transformations.

For the inductive step $k \rightarrow k + 1$, let \mathbb{X} be a $(k + 1)$ -dimensional subspace. We choose an orthonormal basis $\mathbf{x}_1, \dots, \mathbf{x}_{k+1}$ of \mathbb{X} such that \mathbf{x}_{k+1} is orthogonal to $\mathbf{u}_1, \dots, \mathbf{u}_k$.

By the induction hypothesis, $\mathbb{U}_k = \text{span}(\mathbf{u}_1, \dots, \mathbf{u}_k)$ is a best-fit k -dimensional subspace. Thus

$$\sum_{i=1}^n \sum_{j=1}^k \langle \mathbf{a}_i, \mathbf{u}_j \rangle^2 \geq \sum_{i=1}^n \sum_{j=1}^k \langle \mathbf{a}_i, \mathbf{x}_j \rangle^2.$$

Moreover, since $\mathbf{x}_{k+1} \perp \mathbf{u}_1, \dots, \mathbf{u}_k$, by the properties of a PCA-transformation we have

$$\sum_{i=1}^n \langle \mathbf{a}_i, \mathbf{u}_{k+1} \rangle^2 \geq \sum_{i=1}^n \langle \mathbf{a}_i, \mathbf{x}_{k+1} \rangle^2.$$

Thus

$$\sum_{i=1}^n \sum_{j=1}^{k+1} \langle \mathbf{a}_i, \mathbf{u}_j \rangle^2 \geq \sum_{i=1}^n \sum_{j=1}^{k+1} \langle \mathbf{a}_i, \mathbf{x}_j \rangle^2.$$

This shows that \mathbb{U}_{k+1} fits the data at least as good as \mathbb{X} , and as \mathbb{X} was arbitrary, this proves the theorem. □

The Covariance Matrix

Let $A \in \mathbb{R}^{n \times \ell}$ be our data matrix. The **covariance matrix** of our data is

$$C := A^\top A \in \mathbb{R}^{\ell \times \ell}.$$

Lemma 5.30

C is symmetric and **positive semi-definite**, that is, all its eigenvalues are nonnegative real numbers.

It follows from the Spectral Decomposition Theorem that \mathbb{R}^ℓ has an orthonormal basis consisting of eigenvectors of C .

Proof of the lemma.

C is obviously symmetric. Let (λ, \mathbf{u}) be an eigenpair of C . Then

$$\lambda \|\mathbf{u}\|^2 = \mathbf{u}^\top \cdot \lambda \cdot \mathbf{u} = \mathbf{u}^\top C \mathbf{u} = (\mathbf{u}^\top A^\top)(A\mathbf{u}) = \langle A\mathbf{u}, A\mathbf{u} \rangle = \|A\mathbf{u}\|^2.$$

Thus $\lambda = \frac{\|A\mathbf{u}\|^2}{\|\mathbf{u}\|^2} \in \mathbb{R}_{\geq 0}$.

□

PCA via Spectral Decomposition of the Covariance Matrix

Theorem 5.31

Let $A \in \mathbb{R}^{n \times \ell}$ be a data matrix and $C = A^\top A$ the corresponding covariance matrix.

Let $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_\ell$ be the eigenvalues of C . Let $\mathbf{u}_1, \dots, \mathbf{u}_\ell$ be an orthonormal basis of \mathbb{R}^ℓ such that \mathbf{u}_j is an eigenvector of C associated with λ_j , and let $U \in \mathbb{R}^{\ell \times \ell}$ be the matrix with columns $\mathbf{u}_1, \dots, \mathbf{u}_\ell$.

Then U is a PCA-transformation of A .

Proof.

Let $k \in [\ell]$. We need to prove that

$$\mathbf{u}_k = \arg \max_{\substack{x \in \mathbb{R}^\ell \\ \text{such that } \|x\|=1 \\ \text{and } x \perp u_1, \dots, u_{k-1}}} \sum_{i=1}^n \langle \mathbf{a}_i, x \rangle^2 = \arg \max_{\substack{x \in \mathbb{R}^\ell \\ \text{such that } \|x\|=1 \\ \text{and } x \perp u_1, \dots, u_{k-1}}} \|Ax\|^2,$$

that is, for all $\mathbf{x} \in \mathbb{R}^\ell$ with $\|\mathbf{x}\| = 1$ and $\mathbf{x} \perp \mathbf{u}_1, \dots, \mathbf{u}_{k-1}$ we have

$$\|A\mathbf{u}_k\|^2 \geq \|Ax\|^2.$$

Let $\mathbf{x} \in \mathbb{R}^\ell$ with $\|\mathbf{x}\| = 1$ and $\mathbf{x} \perp \mathbf{u}_1, \dots, \mathbf{u}_{k-1}$. We write \mathbf{x} in the orthonormal basis $\mathbf{u}_1, \dots, \mathbf{u}_\ell$:

$$\mathbf{x} = \sum_{j=1}^{\ell} x_j \mathbf{u}_j,$$

where $x_j = \langle \mathbf{x}, \mathbf{u}_j \rangle$. As $\mathbf{x} \perp \mathbf{u}_1, \dots, \mathbf{u}_{k-1}$, we have $x_1, \dots, x_{k-1} = 0$ and thus $\mathbf{x} = \sum_{j=k}^{\ell} x_j \mathbf{u}_j$. As $\|\mathbf{x}\| = 1$ we have $\sum_{j=k}^{\ell} x_j^2 = \sum_{j=1}^{\ell} x_j^2 = \|\mathbf{x}\| = 1$.

It follows that

$$\begin{aligned}\|A\mathbf{x}\|^2 &= \left\| \sum_{j=k}^{\ell} x_j A\mathbf{u}_j \right\|^2 \\&= \left\langle \sum_{j=k}^{\ell} x_j A\mathbf{u}_j, \sum_{j=k}^{\ell} x_j A\mathbf{u}_j \right\rangle \\&= \sum_{j=k}^{\ell} \sum_{j'=k}^{\ell} x_j x'_j \langle A\mathbf{u}_j, A\mathbf{u}_{j'} \rangle \\&= \sum_{j=k}^{\ell} \sum_{j'=k}^{\ell} x_j x_{j'} \cdot \mathbf{u}_j^\top C \mathbf{u}_{j'} \\&= \sum_{j=k}^{\ell} \sum_{j'=k}^{\ell} x_j x_{j'} \lambda_{j'} \langle \mathbf{u}_j, \mathbf{u}_{j'} \rangle && \text{because } C\mathbf{u}_{j'} = \lambda_{j'} \mathbf{u}_{j'} \\&= \sum_{j=k}^{\ell} x_j^2 \lambda_j && \text{because } \mathbf{u}_1, \dots, \mathbf{u}_\ell \text{ is an ONB} \\&\leq \lambda_k \sum_{j=k}^{\ell} x_j^2 && \text{because } \lambda_k \geq \lambda_j \text{ for } j \geq k\end{aligned}$$

because $\sum_{j=k}^{\ell} x_j^2 = 1$

$$\begin{aligned}&= \lambda_k \\&= \langle \mathbf{u}_k, \lambda_k \mathbf{u}_k \rangle \\&= \langle \mathbf{u}_k, C \mathbf{u}_k \rangle \\&= \langle A \mathbf{u}_k, A \mathbf{u}_k \rangle \\&= \|A \mathbf{u}_k\|^2.\end{aligned}$$



5.6 Spectral Clustering

The intuitive objective of a clustering algorithm is to partition the data points into clusters in such a way that:

- (i) the points within each cluster are similar;
- (ii) the points in distinct clusters are dissimilar.

Clustering algorithms like k -means focus on (i).

Spectral clustering focusses on (ii).

We have a set X of data points and a **similarity measure** on X , that is, a symmetric function

$$s : X \times X \rightarrow \mathbb{R}_{\geq 0}$$

Symmetric means that $s(x, y) = s(y, x)$ for all $x, y \in X$.

Example 5.32

If $X \subseteq \mathbb{R}^\ell$, we can let $s(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2}$.

Then $0 < s(\mathbf{x}, \mathbf{y}) \leq 1$, with equality if and only if $\mathbf{x} = \mathbf{y}$.

Objective

Partition X into k (nonempty) clusters C^1, \dots, C^k in a way that minimises the overall similarity between points in distinct clusters.

For this problem, it does not really matter what the elements of X are, all that matters is their similarity values. Therefore, we simply assume $X = [n]$, and we define the **similarity matrix** $S \in \mathbb{R}^{n \times n}$ by $S_{ij} := s(i, j)$.

Clustering Based on Minimum Cuts

Idea

Choose a partition C^1, \dots, C^k that minimises

$$\text{mincut}(C^1, \dots, C^k) := \sum_{p=1}^k \sum_{i \in C^p, j \notin C^p} S_{ij}.$$

Disadvantage

Minimum cuts favour very small (or very large) clusters; they are often obtained by choosing all clusters but one of size 1. This is typically not what we want.

Clustering Based on Balanced Cuts

Idea

Choose a partition C^1, \dots, C^k that minimises

$$\text{balcut}(C^1, \dots, C^k) := \sum_{p=1}^k \frac{1}{|C^p|} \sum_{i \in C^p, j \notin C^p} S_{ij}.$$

Difficulty

Minimising the `balcut` function is computationally hard.

The Laplacian

Definition 5.33

The **Laplacian** of S is the matrix

$$L = D - S \in \mathbb{R}^{n \times n},$$

where D is the diagonal matrix with entries $D_{ii} = \sum_{j=1}^n S_{ij}$.

Lemma 5.34

For every vector $\mathbf{v} = (v_1, \dots, v_n)^\top$ we have

$$\mathbf{v}^\top L \mathbf{v} = \frac{1}{2} \sum_{i,j} S_{ij} (v_i - v_j)^2.$$

Corollary 5.35

The Laplacian is positive semi-definite.

Proof of Lemma 5.34.

$$\begin{aligned}\sum_{i,j} S_{ij}(v_i - v_j)^2 &= \sum_{i,j} S_{ij}v_i^2 - 2\sum_{i,j} S_{ij}v_i v_j + \sum_{i,j} S_{ij}v_j^2 \\&= 2\left(\sum_i v_i^2 \sum_j S_{ij} - \sum_{i,j} v_i v_j S_{ij}\right) \\&= 2\left(\sum_i v_i^2 D_{ii} - \sum_{i,j} v_i v_j S_{ij}\right) \\&= 2(\mathbf{v}^\top D \mathbf{v} - \mathbf{v}^\top S \mathbf{v}) \\&= 2\mathbf{v}^\top L \mathbf{v}.\end{aligned}$$

□

Proof of Corollary 5.34.

The Laplacian is a real symmetric matrix, thus by the spectral decomposition there is an orthogonal matrix U such that

$$U^\top L U = \Lambda,$$

where Λ is the diagonal matrix whose diagonal entries are the eigenvalues of L . Hence each eigenvalue λ_i can be written as $\mathbf{u}_i^\top L \mathbf{u}_i$, where \mathbf{u}_i is the i th column of U . By Lemma 5.34, $\mathbf{u}_i^\top L \mathbf{u}_i$ is nonnegative. □

Connection with Cuts

Lemma 5.36

Let C^1, \dots, C^k be a partition of $[n]$, and let $U \in \mathbb{R}^{n \times k}$ be the matrix with entries

$$U_{ip} := \begin{cases} \frac{1}{\sqrt{|C^p|}} & \text{if } i \in C^p, \\ 0 & \text{otherwise} \end{cases}$$

Then

$$\text{balcut}(C^1, \dots, C^k) = \text{trace}(U^\top L U).$$

Proof.

Let $\mathbf{u}_1, \dots, \mathbf{u}_k \in \mathbb{R}^n$ be the columns of U , where $\mathbf{u}_p = (u_{1p}, \dots, u_{np})^\top$.

Observe that U is **orthonormal**, that is, $\|\mathbf{u}_p\| = 1$ for all p and $\langle \mathbf{u}_p, \mathbf{u}_q \rangle = 0$ for $p \neq q$.

Note that for all $p \in [k]$, $i, j \in [n]$ we have

$$(u_{ip} - u_{jp})^2 = \begin{cases} 0 & \text{if } \mathbf{x}_i \in C_p \text{ and } \mathbf{x}_j \in C_p, \\ \frac{1}{|C^p|} & \text{if } \mathbf{x}_i \in C_p \text{ and } \mathbf{x}_j \notin C_p, \\ \frac{1}{|C^p|} & \text{if } \mathbf{x}_i \notin C_p \text{ and } \mathbf{x}_j \in C_p \\ 0 & \text{if } \mathbf{x}_i \notin C_p \text{ and } \mathbf{x}_j \notin C_p. \end{cases}$$

Thus by Lemma 5.34 we have

$$\begin{aligned} \mathbf{u}_p^\top L \mathbf{u}_p &= \frac{1}{2} \sum_{i,j} S_{ij} (u_{ip} - u_{jp})^2 \\ &= \frac{1}{2} \sum_{i \in C^p, j \notin C^p} \frac{S_{ij}}{|C^p|} + \frac{1}{2} \sum_{i \notin C^p, j \in C^p} \frac{S_{ij}}{|C^p|} \end{aligned}$$

$$\begin{aligned}
&= \sum_{i \in C^p, j \notin C^p} \frac{S_{ij}}{|C^p|} \\
&= \frac{1}{|C^p|} \sum_{i \in C^p, j \in [n] \setminus C^p} S_{ij}.
\end{aligned}$$

It follows that

$$\begin{aligned}
\text{trace}(U^\top L U) &= \sum_{p=1}^k \mathbf{u}_p^\top L \mathbf{u}_p \\
&= \sum_{p=1}^k \frac{1}{|C^p|} \sum_{i \in C^p, j \notin C^p} S_{ij} \\
&= \text{balcut}(C^1, \dots, C^k).
\end{aligned}$$

□

A Generalisation

Let us call a matrix $U \in \mathbb{R}^{n \times k}$ a **partition matrix** if U has exactly k distinct rows.

That is, that is, there is a partition C^1, \dots, C^k of $[n]$ such that for $i, j \in C^p$, rows i and j are equal, and for $i \in C^p, j \in C^q$ with $p \neq q$ rows i and j are distinct.

We call C^1, \dots, C^k the **partition associated with U** .

Example 5.37

The matrix U of Lemma 5.36 is an orthonormal partition matrix.

Lemma 5.38

Let $U \in \mathbb{R}^{n \times k}$ be an orthonormal partition matrix, and let C^1, \dots, C^k be the partition of $[n]$ associated with U . Then

$$\text{balcut}(C^1, \dots, C^k) = \text{trace}(U^\top L U).$$

Proof.

For every $p \in k$, let $\mathbf{u}_p = (u_{1p}, \dots, u_{np})^\top$ be the p th column of U , and for every $i \in [n]$, let $\mathbf{x}_i = (u_{i1}, \dots, u_{ik})$ be the the i th row of U .

For every $p \in [k]$, let $\mathbf{x}^p := \mathbf{x}_i$ for some (and hence for all) $i \in C^p$. Let $c_p := |C^p|$ and $\tilde{\mathbf{x}}^p := \sqrt{c_p} \mathbf{x}^p$.

Let $\tilde{U} \in \mathbb{R}^{k \times k}$ be the matrix with rows $\tilde{\mathbf{x}}^1, \dots, \tilde{\mathbf{x}}^k$.

Claim 1

\tilde{U} is an orthogonal matrix.

Proof.

Let $\tilde{\mathbf{u}}_1, \dots, \tilde{\mathbf{u}}_k$ be the columns of \tilde{U} . Observe that for all $p, q \in [k]$ it holds that $\langle \tilde{\mathbf{u}}_i, \tilde{\mathbf{u}}_j \rangle = \langle \mathbf{u}_i, \mathbf{u}_j \rangle$. As U is orthonormal, it follows that \tilde{U} is orthonormal as well, and as it is a square matrix, it follows that it is orthogonal. \square

It follows that $\tilde{\mathbf{x}}^1, \dots, \tilde{\mathbf{x}}^k$ are orthonormal, and thus

$$\|\mathbf{x}^p\|^2 = \frac{1}{c_p} \|\tilde{\mathbf{x}}^p\|^2 = \frac{1}{c_p} \quad \text{for all } p \in [k]$$

and $\langle \mathbf{x}_p, \mathbf{x}_q \rangle = 0$ for all $p \neq q$. By Pythagoras' Theorem, it follows that

$$\|\mathbf{x}^p - \mathbf{x}^q\|^2 = \frac{1}{c_p} + \frac{1}{c_q}.$$

By Lemma 5.34 we have

$$\begin{aligned}\text{trace}(U^\top L U) &= \sum_{p=1}^k \mathbf{u}_p^\top L \mathbf{u}_p \\ &= \sum_{p=1}^k \left(\frac{1}{2} \sum_{i,j} S_{ij} (u_{ip} - u_{jp})^2 \right) \\ &= \frac{1}{2} \sum_{i,j} S_{ij} \sum_{p=1}^k (u_{ip} - u_{jp})^2 \\ &= \frac{1}{2} \sum_{i,j} S_{ij} \|\mathbf{x}_i - \mathbf{x}_j\|^2 \\ &= \frac{1}{2} \sum_{\substack{p,q \in [k] \\ p \neq q}} \|\mathbf{x}^p - \mathbf{x}^q\|^2 \sum_{\substack{i \in C^p \\ j \in C^q}} S_{ij}\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{2} \sum_{\substack{p,q \in [k] \\ p \neq q}} \left(\frac{1}{c_p} + \frac{1}{c_q} \right) \sum_{i \in C^p, j \in C^q} S_{ij} \\
&= \frac{1}{2} \sum_{p \in [k]} \frac{1}{c_p} \sum_{\substack{q \in [k] \\ p \neq q}} \sum_{i \in C^p, j \in C^q} S_{ij} + \frac{1}{2} \sum_{q \in [k]} \frac{1}{c_q} \sum_{\substack{p \in [k] \\ p \neq q}} \sum_{i \in C^p, j \in C^q} S_{ij} \\
&= \sum_{p \in [k]} \frac{1}{c_p} \sum_{\substack{q \in [k] \\ p \neq q}} \sum_{i \in C^p, j \in C^q} S_{ij} \\
&= \sum_{p \in [k]} \frac{1}{c_p} \sum_{i \in C^p, j \notin C^p} S_{ij} \\
&= \text{balcut}(C^1, \dots, C^k).
\end{aligned}$$

□

Corollary 5.39

Finding a partition C^1, \dots, C^k that minimises $\text{balcut}(C^1, \dots, C^k)$ is equivalent to finding an orthonormal partition matrix $U \in \mathbb{R}^{n \times k}$ that minimises $\text{trace}(U^\top L U)$.

The difficulty remains: finding such a partition matrix is computationally hard.

Idea

Instead of a partition matrix, compute an arbitrary orthonormal matrix U . This is feasible.

Theorem 5.40

Let $U \in \mathbb{R}^{n \times k}$ be an orthonormal matrix whose columns are eigenvectors to the k smallest eigenvalues of L . Then

$$\text{trace}(U^\top LU) \leq \text{trace}(V^\top LV)$$

for all orthonormal matrices $V \in \mathbb{R}^{n \times k}$.

Idea (cont'd)

Then try to find a partition matrix close to the orthonormal matrix U and return the associated partition.

Notes for Page 5.53

The theorem can be proved similarly to the results on PCA in the previous section.

Spectral Clustering Algorithm

Algorithm SPECTRAL CLUSTERING

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters

1. Compute Laplacian L of S
2. Compute orthonormal matrix $U \in \mathbb{R}^{n \times k}$ whose columns are eigenvectors to the k smallest eigenvalues of L
3. let $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^k$ be the rows of U
4. cluster $\mathbf{x}_1, \dots, \mathbf{x}_n$ using k -means, yielding clusters C^1, \dots, C^k
5. **return** C^1, \dots, C^k

Notes for Page 5.54

The intuition is that if the vectors in each of the clusters C^p are close together, then we can find a partition matrix \hat{U} with associated partition C^1, \dots, C^k such that \hat{U} is close to U and hence $\hat{U}^\top L \hat{U}$ is close to $U^\top L U$.

- ▶ Spectral clustering works well in practice, in particular in situations where the points within each cluster are not particularly close together (for example, they may be stretched along a line or a circle), but the clusters are well-separated.
- ▶ The version of spectral clustering we introduced here is the simplest. There are other versions, for example one that uses a normalised form of the Laplacian matrix.

Most of the material of Sections 5.1–5.5 is covered in [Shalev-Shwartz and Ben-David, 2014, Chapter 23]. The Perron Frobenius Theorem (Theorem 5.21) and Theorem 5.23 can be found in [Horn and Johnson, 2012, Chapter 8]. Details for Sections 5.1 and 5.2 can be found in [Blum et al., 2020, Chapter 1]. [Leskovec et al., 2014, Chapter 11] gives a high level overview of the material of Sections 5.3–5.5. Our account of spectral clustering follows [Shalev-Shwartz and Ben-David, 2014, Section 22.3]. An excellent survey that delves deeper into spectral clustering is [von Luxburg, 2007].



Chapter 6

The Monte Carlo Method

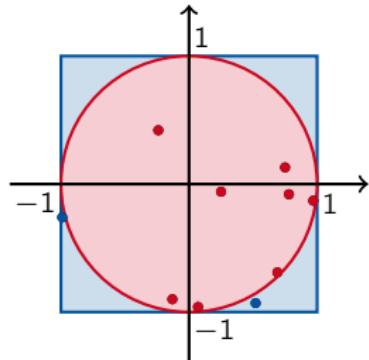
Goals of this Chapter

Monte Carlo methods aim to estimate values through sampling.

- ▶ We will start discussing the method using basic sampling techniques.
- ▶ We will introduce (or review, depending on your knowledge) Markov chains and then introduce the Markov Chain Monte Carlo (MCMC) method.
- ▶ We will discuss the mixing time of Markov chains, which is the crucial parameter determining the efficiency of the MCMC method.

6.1 Estimation through Sampling

Example: Estimating π



Task

Estimate π , the area of a unit disk in the plane.

Method

1. For some m (to be determined), independently sample m points $\mathbf{x}_1, \dots, \mathbf{x}_m$ uniformly at random from the square $[-1, 1] \times [-1, 1]$.
2. For each i , let $y_i := \begin{cases} 1 & \text{if } \|\mathbf{x}_i\| \leq 1, \\ 0 & \text{otherwise.} \end{cases}$
3. Return $\hat{\pi} := \frac{4}{m} \sum_{i=1}^m y_i$ as the estimate.

Example

In our example of 10 randomly chosen points, $\hat{\pi} = 3.2$

Notes for Page 6.4

Let Y_i be the indicator random variable for the event " $\|\mathbf{x}_i\| \leq 1$ " (that is, " \mathbf{x}_i is in the unit disk") and $Y = \sum_{i=1}^m Y_i$. Note that $\hat{\pi} = \frac{4Y}{m}$.

We have

$$\Pr(Y_i = 1) = \frac{\text{area of the unit disk}}{\text{area of the square } [-1, 1] \times [-1, 1]} = \frac{\pi}{4}.$$

Thus by linearity of expectation,

$$E(Y) = \frac{m\pi}{4}.$$

It follows that $E(\hat{\pi}) = \frac{4E(Y)}{m} = \pi$.

By the Hoeffding bound we have

$$\begin{aligned}\Pr(|\hat{\pi} - \pi| \geq \varepsilon) &= \Pr\left(\left|Y - \frac{m\pi}{4}\right| \geq \frac{\varepsilon}{4}m\right) \\ &= \Pr\left(|Y - E(Y)| \geq \frac{\varepsilon}{4}m\right) \leq 2 \exp\left(-\frac{\varepsilon^2}{8}m\right)\end{aligned}$$

For example, if we sample $m = 5000$ points then with probability ≥ 0.99 our estimate is correct up to an additive error of 0.1.

Goal

Estimate a quantity μ .

Method

- ▶ Express μ as the expected value of a random variable f defined on a (potentially very large) sample space Ω .
- ▶ Draw m independent samples $\omega_1, \dots, \omega_m \in \Omega$, where m depends on the desired error bound ε and confidence bound δ .
- ▶ Return $\hat{\mu} = \frac{1}{m} \sum_{i=1}^m f(\omega_i)$ as the estimate.

Lemma 6.1

Let $\varepsilon, \delta \in \mathbb{R}_{>0}$, and let $b := \max \{|f(\omega)| \mid \omega \in \Omega\}$. If $m \geq \frac{b \ln(\frac{2}{\delta})}{\varepsilon^2}$ then

$$\Pr(|\hat{\mu} - \mu| \leq \varepsilon) \geq 1 - \delta.$$

To prove the lemma, we need a slightly more general version of the Hoeffding Bounds, which we state without proof.

Lemma 6.2

Let $a, b \in \mathbb{R}$ with $a < b$. Let X_1, \dots, X_n be a sequence of independent identically distributed random variables with $\text{rg}(X_i) \subseteq [a, b]$, and let

$X := \sum_{i=1}^n X_i$. Then for all $\varepsilon > 0$:

$$\Pr\left(\left|X - \mathbb{E}(X)\right| \geq \varepsilon n\right) \leq 2 \exp\left(-\frac{2\varepsilon^2}{b-a}n\right).$$

Proof of Lemma 6.1.

Let $X_i := f(\omega_i)$ and $X = \sum_{i=1}^m X_i$. Note that

$$\mu = \mathbb{E}(f) = \mathbb{E}(X_i) = \frac{1}{m} \mathbb{E}(X).$$

Since $\text{rg}(X_i) \in [-b, b]$, by Lemma 6.2 we have

$$\Pr(|\hat{\mu} - \mu| \geq \varepsilon) = \Pr(|X - m\mu| \geq \varepsilon m) \leq 2 \exp\left(-\frac{\varepsilon^2}{b}m\right).$$

With $m \geq \frac{b \ln(2/\delta)}{\varepsilon^2}$, we have

$$\frac{\varepsilon^2}{b}m \geq \ln\left(\frac{2}{\delta}\right)$$

and thus

$$2 \exp\left(-\frac{\varepsilon^2}{b}m\right) \leq 2 \exp\left(-\ln\left(\frac{2}{\delta}\right)\right) = \delta.$$

The assertion of the lemma follows. □

Notation

For a probability distribution \mathcal{P} on a finite sample space Ω and an $x \in \Omega$, we write $\mathcal{P}(x)$ instead of $\mathcal{P}(\{x\})$.

Example: Sampling Water from a Lake

Suppose we want to monitor the water quality in a lake by estimating key parameters like oxygen content or pH value.

Task

Draw a water sample of 1cm^3 uniformly from a lake.

We see the surface S of the lake and can sample points from S uniformly, and at each point x of the surface we can measure the depth $\mathcal{D}(x)$.

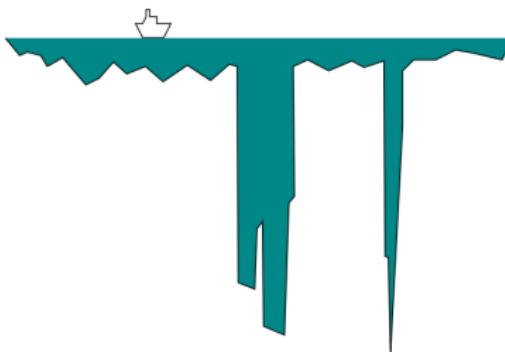


Figure from [MacKay, 2003]

We assign coordinates to the points in the lake: (x, y) is the point a depth $y \in [0, \mathcal{D}(x)]$ below point $x \in S$.

Example (cont'd)

Approach 1

Uniformly sample a point $x \in S$ and then uniformly a $y \in [0, \mathcal{D}(x)]$ and draw the water sample at (x, y) .

Problem: The sample is not uniform, because the points in deeper regions are less likely to be hit.

Approach 2

1. Sample points $x_1, \dots, x_n \in S$ uniformly and measure their depths $d_i := \mathcal{D}(x_i)$.
2. Randomly choose an $i \in [n]$ with probability $p_i := \frac{d_i}{\sum_{j=1}^n d_j}$.
3. Choose $y_i \in [0, d_i]$ uniformly at random and draw the water sample at (x_i, y_i) .

Problem: Unless we choose n to be very large, we may miss points on the surface that are above very narrow and deep canyons. Thus our sample is still biased.

Example (cont'd)

For the next approach, we assume that we know an upper bound Δ on the depth of the lake.

Approach 3 (Rejection Sampling)

1. Uniformly sample a point $x \in S$.
2. Uniformly sample $y \in [0, \Delta]$.
3. If $y \leq \mathcal{D}(x)$, draw the water sample at (x, y) ; otherwise reject (x, y) and repeat (go back to step 1).

This method generates a uniform sample.

Problem: If most places in the lake are quite shallow, and there are only a few very deep canyons, or if the upper bound Δ on the depth is too pessimistic, then the method is inefficient because it rejects too often.

Proof that the sample is uniform.

We only give a proof for a finite version of the problem, where we assume that we have partitioned the lake into small cubes of size 1cm^3 .

Then the surface S is a finite set, and for $x \in S$ the cubes below x are (x, y) for integers $y \in [\mathcal{D}(x)]$. Thus the sample space is

$$\Omega = \left\{ (x, y) \mid x \in S, y \in [\mathcal{D}(x)] \right\},$$

For every $(x, y) \in \Omega$, the probability that (x, y) is returned in an iteration of the algorithm is

$$\frac{1}{|S|} \cdot \frac{\mathcal{D}(x)}{\Delta} \cdot \frac{1}{\mathcal{D}(x)} = \frac{1}{|S| \cdot \Delta}.$$

Here we think of the process as first deciding, with probability $\mathcal{D}(x)/\Delta$, whether to continue or reject, and then if we decide to continue, we sample y uniformly from $\mathcal{D}(x)$.

Note that the probability is independent of (x, y) . Thus the algorithm samples uniformly.

In any round the algorithm rejects with some probability $p < 1$. Thus the probability that the algorithm has not stopped after k rounds is p^k , and as $p^k \xrightarrow{k \rightarrow \infty} 0$, this means that with probability 1 the algorithm generates a sample eventually. □

Remark

The rejection sampling method can be improved: instead of choosing one Δ such that $\Delta \geq \mathcal{D}(x)$ for all x , we can also work with a function $\Delta : S \rightarrow \mathbb{R}$ such that $\Delta(x) \geq \mathcal{D}(x)$ for all x .

This may help if we already have an idea of the depth profile of the lake.

Discussion

- ▶ We are not particularly interested in sampling water from a lake, but the example serves as a very nice analogy.
- ▶ The problem we actually need to solve in the example is to randomly sample a point x from the surface S of the lake with probability $\mathcal{P}(x)$ proportional to $\mathcal{D}(x)$:

$$\mathcal{P}(x) = \frac{\mathcal{D}(x)}{Z} \quad \text{with} \quad Z = \sum_x \mathcal{D}(x).$$

(In the continuous setting of our lake example, we should rather think of \mathcal{P} as a probability density function and replace the sum by an integral.)

- ▶ One problem is that we do not know the normalising constant Z .

Formal Framework

Suppose we want to sample from a probability space (Ω, \mathcal{P}) .

- ▶ In our formal treatment, we always assume that Ω is finite. However, it is typically too large to be enumerated efficiently.
- ▶ We assume that we know the sample space Ω and have a representation of its elements.
- ▶ We further assume that we know the probability distribution (or density) \mathcal{P} up to a normalising constant. That is, given an element $\omega \in \Omega$, we can efficiently compute

$$\mathcal{D}(\omega) := Z \cdot \mathcal{P}(\omega),$$

where $Z > 0$ is an unknown constant.

Remark 6.3

Some examples involve infinite spaces. We simply assume that we discretise them (as we did in the lake example). But the theory can also be extended to infinite probability spaces with continuous distributions.

Rejection Sampling

As before, we want to sample from (Ω, \mathcal{P}) where $\mathcal{P}(\omega) = \frac{\mathcal{D}(\omega)}{Z}$.

- We assume that we know a $\Delta : \Omega \rightarrow \mathbb{R}$ such that

$$\mathcal{D}(\omega) \leq \Delta(\omega) \quad \text{for all } \omega \in \Omega.$$

- Furthermore, we assume that we can sample elements from Ω with probability

$$\Pi(\omega) = \frac{\Delta(\omega)}{\sum_{\omega' \in \Omega} \Delta(\omega')}$$

proportional to $\Delta(\omega)$. We call Π the **proposal distribution**.

- Often, Δ is just a constant function. Then Π is the uniform distribution on Ω .

Rejection Sampling

1. Sample a point $\omega \in \Omega$ from distribution Π
2. With probability $\frac{\mathcal{D}(\omega)}{\Delta(\omega)}$, **accept** and return ω ; otherwise **reject** (and repeat).

Lemma 6.4

- (1) Let a be the probability that rejection sampling accepts in the first iteration (or any other iteration). Then

$$a = \frac{Z}{\sum_{\omega \in \Omega} \Delta(\omega)}.$$

- (2) Rejection sampling eventually accepts with probability 1, and it accepts in at most k iterations with probability at least

$$1 - e^{-ak}$$

- (3) If rejection sampling accepts, then it returns each $\omega \in \Omega$ with probability $\mathcal{P}(\omega)$.

In the proof, we write RS to abbreviate rejection sampling.

Proof of Lemma 6.4.

Assertion (1) is proved by straightforward calculation:

$$a = \sum_{\omega \in \Omega} \Pi(\omega) \cdot \frac{\mathcal{D}(\omega)}{\Delta(\omega)} = \sum_{\omega \in \Omega} \frac{\mathcal{D}(\omega)}{\sum_{\omega' \in \Omega} \Delta(\omega')} = \frac{Z}{\sum_{\omega \in \Omega} \Delta(\omega)}.$$

To prove (2), note that the probability a_k that RS accepts in k iterations is the probability of a success after k independent Bernoulli trials with success probability a , which is geometrically distributed. Thus

$$a_k = (1 - a)^{k-1} a.$$

Hence the probability that RS never accepts is

$$1 - \sum_{k \geq 1} (1 - a)^{k-1} a = 1 - a \sum_{k \geq 0} (1 - a)^k = 0.$$

Here we use that $x \in (0, 1)$, $n \in \mathbb{N}_{>0}$ we have $\sum_{k \geq 0} x^k = \frac{1}{1-x}$.

Furthermore, the probability that RS accepts in at most k rounds is

$$\begin{aligned} \sum_{i=1}^k (1-a)^{i-1} a &= a \sum_{i=0}^{k-1} (1-a)^i \\ &= a \frac{1 - (1-a)^k}{1 - (1-a)} = 1 - (1-a)^k \\ &\geq 1 - e^{-ak}. \end{aligned}$$

Here we use that for all $x \in \mathbb{R} \setminus \{0\}$, $n \in \mathbb{N}_{>0}$ we have $\sum_{i=0}^{n-1} x^i = \frac{1-x^n}{1-x}$ and $(1-x)^n \leq e^{-xn}$.

To prove (3), for every $k \in \mathbb{N}_{>0}$, let r_k be the probability that RS rejects in the first k iterations. Moreover, let $r_0 := 1$. Then for every $k \in \mathbb{N}_{>0}$ and $\omega \in \Omega$, the probability that ω is returned in round k is

$$r_{k-1} \cdot \Pi(\omega) \cdot \frac{\mathcal{D}(\omega)}{\Delta(\omega)} = \frac{r_{k-1} \cdot Z}{\sum_{\omega' \in \Omega} \Delta(\omega')} \cdot \mathcal{P}(\omega).$$

Thus the probability that RS returns ω is

$$\mathcal{P}(\omega) \sum_{k \geq 1} \frac{r_{k-1} \cdot Z}{\sum_{\omega' \in \Omega} \Delta(\omega')}.$$

As this gives is a probability distribution on Ω and the sum does not depend on ω , we must have $\sum_{k \geq 1} \frac{r_{k-1} \cdot Z}{\sum_{\omega' \in \Omega} \Delta(\omega')} = 1$, which proves assertion (3). \square

Example: Sampling in Euclidean Space

We want to sample points uniformly from a (measurable) $X \subseteq \mathbb{R}^\ell$. We assume that

- ▶ We have an efficient membership test for X .
- ▶ We know a (large) cuboid $Q \subseteq \mathbb{R}^\ell$ such that $X \subseteq Q$.
- ▶ We have discretised Q (partitioned into small cubes) such that X is a union of cubes.

Our universe is $\Omega := Q$, and the probability distribution \mathcal{P} is the uniform distribution on X . We $\mathcal{D} := \mathbb{1}_X$, that is, $\mathcal{D}(x) = 1$ if $x \in X$ and $\mathcal{D}(x) = 0$. otherwise.

We apply rejection sampling with the uniform distribution on Q as proposal distribution Π . We let $\Delta(x) := 1$ for all $x \in Q$.

Observe that $\mathcal{D}(x) \leq \Delta(x)$ for all $x \in \Omega$ and $\Pi = \frac{\Delta}{\text{vol}(Q)}$.

- ▶ Observe that the normalising constant Z is the volume of X :

$$Z = \int_Q \mathcal{D}(\mathbf{x}) d\mathbf{x} = \int_X \mathbf{1} d\mathbf{x} = \text{vol}(X).$$

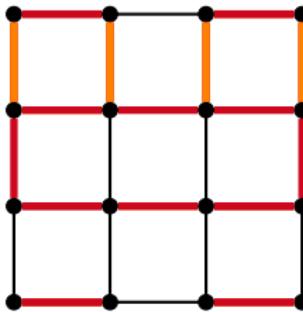
- ▶ Thus we can also use this approach to estimate the volume $\text{vol}(X)$: the acceptance probability is $\frac{Z}{\text{vol}(Q)}$, and since we know $\text{vol}(Q)$, this gives us an estimator for $Z = \text{vol}(X)$.
- ▶ A problem with this approach is that in high dimensional space, the acceptance probability $\frac{Z}{\sum_{\omega \in \Omega} \Delta(\omega)} = \frac{\text{vol}(X)}{\text{vol}(Q)}$ tends to get very small.

Example: Sampling Matchings

Recall that a **matching** of a graph G is a set $M \subseteq E(G)$ of mutually disjoint edges of G (that is, no two edges in M share an endvertex). A matching M is **perfect** if every vertex of G is an endvertex of an edge in M .

Example 6.5

matchings in a
 (4×4) -grid



Task

Given a graph G , uniformly sample a matching of G .

Variants

- (1) Sample matchings M with probability $\frac{\lambda^{|M|}}{Z_\lambda}$ for an arbitrary $\lambda > 0$ and a suitable normalising constant Z_λ .

This is related to a model of statistical physics known as the **monomer-dimer model**.

- (2) Uniformly sample perfect matchings M (assuming there is at least one perfect matching).

This is related to computing the **permanent** of a nonnegative matrix.

Sampling Matchings with Rejection Sampling

Let $G = (V, E)$ be a graph and $\mathcal{M} \subseteq 2^E$ be the set of all matchings of G .

- ▶ We let $\Omega := 2^E$ and let $\mathcal{D} := \mathbb{1}_{\mathcal{M}}$.
Then $\mathcal{P} := \frac{\mathcal{D}}{Z}$, where $Z = |\mathcal{M}|$, is the uniform distribution on \mathcal{M} .
- ▶ We let $\Delta := \mathbb{1}_{2^E}$. Then the proposal distribution is the uniform distribution on 2^E .
The acceptance probability is $\frac{|\mathcal{M}|}{2^E}$, thus we can use this approach to estimate the number of matchings.

Problem

The acceptance probability $\frac{|\mathcal{M}|}{2^E}$ is usually very small, exponentially small in the size of the graph for most graphs, so this approach is not practical.

Example: Sampling Satisfying Assignments

Task

Uniformly sample satisfying assignments for a given Boolean formula φ .

Rejection Sampling Approach

Sample arbitrary assignments and reject if they are not satisfying.

Problem

- ▶ Unless φ has many satisfying assignments, the rejection sampling approach is not practical, because the acceptance probability is too low.
- ▶ It is not difficult to prove that, unless $P = NP$, there is no polynomial time algorithm uniformly (or almost uniformly) sampling satisfying assignments of a given (satisfiable) formula in conjunctive normal form.

The Karp-Luby Algorithm

Theorem 6.6

There is a randomised polynomial time algorithm that, given a satisfiable Boolean formula φ in disjunctive normal form, returns a satisfying assignment sampled uniformly at random from the set of all satisfying assignments.

Remark 6.7

The same algorithm can also be used to obtain an estimate on the number of satisfying assignments.

Proof of Theorem 6.6.

The trick is to apply rejection sampling in a different than the obvious setting. Let $V := \text{var}(\varphi)$ be the set of all variables of φ . Let $n := |V|$ and suppose that

$$\varphi = \bigvee_{i=1}^m \gamma_i,$$

where

$$\gamma_i = \bigwedge_{j=1}^{k_i} \lambda_{ij}$$

with $\lambda_{ij} \in \{X_{ij}, \neg X_{ij}\}$ for some $X_{ij} \in V$.

Without loss of generality, we may assume that in all γ_i each variable occurs at most once, that is, for all $i \in [m]$, $j, j' \in [k_i]$ with $j \neq j'$ we have $\lambda_{ij} \neq \lambda_{ij'}$ and $\lambda_{ij} \neq \neg \lambda_{ij'}$. (If $\lambda_{ij} = \neg \lambda_{ij'}$ then γ_i is unsatisfiable, and we can simply discard it from the disjunction).

Let $A := \{0, 1\}^V$ be the set of all assignments, and let $S \subseteq A$ be the set of all satisfying assignments for φ . Moreover, for $i \in [m]$, let S_i be the set of all satisfying assignments for γ_i .

Claim 1

- (1) $S = \bigcup S_i$;
- (2) $|S_i| = 2^{n-k_i}$.

Proof.

Assertion (1) is trivial. For (2), observe that for an assignment α to satisfy γ_i , the k_i variables in γ_i can only take one variable, and the remaining $n - k_i$ variables can take arbitrary values. \square

Now we are ready to set up our RS framework. We let

$$\Omega := \{(i, \alpha) \mid i \in [m], \alpha \in S_i\}$$

and

$$X := \{(i, \alpha) \in \Omega \mid \alpha \notin S_j \text{ for all } j < i\}.$$

For every $\alpha \in S$, we let

$$\text{minind}(\alpha) := \min \{i \in [m] \mid \alpha \in S_i\}.$$

Note that $\text{minind}(\alpha)$ is well-defined, because every $\alpha \in S$ satisfies some γ_i .

Claim 2

The mapping $h : S \rightarrow X$ defined by $h(\alpha) := (\text{minind}(\alpha), \alpha)$ is bijective.

Proof.

h is well-defined, because minind is, and it is obviously injective. It is surjective, because if $(i, \alpha) \in X$, then $i = \text{minind}(\alpha)$ and thus $(i, \alpha) = h(\alpha)$. \sqcup

We let $\mathcal{D} := \mathbb{1}_X$, so $\mathcal{P} := \frac{\mathcal{D}}{Z}$ with $Z = |X| = |S|$ is the uniform distribution on X . As the proposal distribution Π we take the uniform distribution on Ω , that is, we let $\Delta(\omega) := 1$ for all $\omega \in \Omega$.

Note that we can efficiently sample from Π :

1. We choose $i \in [m]$ uniformly at random.
2. We choose a random $\alpha \in S_i$ by first fixing the variables appearing in γ_i to the values given by γ_i and then randomly choosing values for all remaining variables.

The acceptance probability of our RS is

$$a := \frac{Z}{|\Omega|} = \frac{|S|}{|\Omega|}.$$

Let $i^* := \arg \max_{i \in [m]} |S_i|$ be the index of the γ_i with the largest number of satisfying assignments (that is, the fewest variables). Note that

$$|S| \geq |S_{i^*}|$$

and

$$|\Omega| = \sum_{i=1}^m |S_i| \leq m|S_{i^*}|$$

Thus

$$a = \frac{|S|}{|\Omega|} \geq \frac{|S_{i^*}|}{m|S_{i^*}|} = \frac{1}{m}.$$

This means that in expectation, RS returns a sample in m steps, and hence it runs in expected polynomial time. □

6.2 Random Walks and Markov Chains

Random Walks on Graphs

Let G be a directed graph, possibly with loops.

- ▶ A **random walk** on G consists of a sequence of vertices generated by
 - ▶ starting at some initial vertex,
 - ▶ in each step randomly choosing an edge out of the current vertex to determine the next vertex.
- ▶ In the simplest case, each outgoing edge of the current vertex is chosen with the same probability.
- ▶ More generally, we may have a probability distribution on the outgoing edges. We view the probability that an edge is taken as the weight of that edge.
- ▶ The initial vertex may be fixed, but it may also be chosen according to some probability distribution on the vertex set.

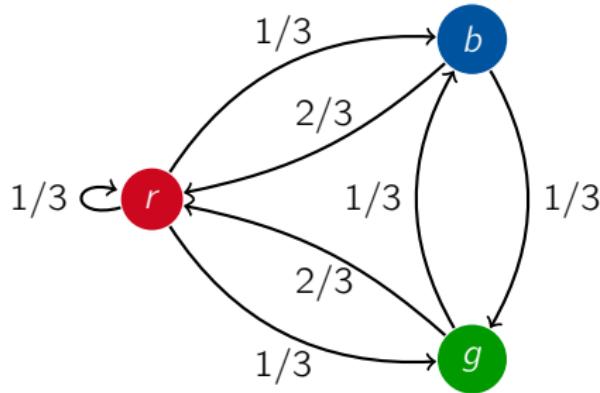
- ▶ A **Markov chain** is a stochastic process that generates a sequence of states from some state space,
 - ▶ starting at some initial state
 - ▶ in each step randomly choosing a new state, based only on the current state and not on the sequence that led to this state.
- ▶ The initial state may be fixed, but it may also be chosen according to some probability distribution on the state space.
- ▶ The state space may be infinite, but in this course it is always finite.
- ▶ A random walk on a graph is a special case where the state space is the vertex set of the graph.
- ▶ Conversely, with each Markov chain we can associate a weighted directed graph with vertex set = state space and edges representing the transitions.

Example 6.8

Consider a Markov chain with three states r , b , and g and the following transition probabilities:

$$\begin{array}{lll} \Pr(r \rightarrow r) = \frac{1}{3} & \Pr(r \rightarrow b) = \frac{1}{3} & \Pr(r \rightarrow g) = \frac{1}{3} \\ \Pr(b \rightarrow r) = \frac{2}{3} & \Pr(b \rightarrow b) = 0 & \Pr(b \rightarrow g) = \frac{1}{3} \\ \Pr(g \rightarrow r) = \frac{2}{3} & \Pr(g \rightarrow b) = \frac{1}{3} & \Pr(g \rightarrow g) = 0. \end{array}$$

We can describe it by the following graph:



Stationary Distribution

Fundamental Property of Markov Chains

Provided the directed graph of a Markov chain is strongly connected, the long-term average probability of the chain being at a particular state is independent of the initial state.

This probability is called the **stationary probability** of the state, and the corresponding probability distribution on the state space is called the **stationary distribution**.

Formal Framework

Let us assume that we have a Markov chain \mathcal{Q} with state space $[n]$, for some $n \in \mathbb{N}$.

Transition Matrix

- ▶ We describe \mathcal{Q} by its **transition (probability) matrix**
 $Q = (q_{ij})_{i,j \in [n]} \in \mathbb{R}^{n \times n}$, where the entry q_{ij} is the probability of a transition from state i to state j .
- ▶ Q is a **stochastic matrix**, that is, all entries are non-negative and all row sums are 1.

Graph and Connectivity

The **(directed) graph** G_Q of a Markov chain with transition matrix $Q = (q_{ij}) \in \mathbb{R}^{n \times n}$ is defined by $V(G_Q) := [n]$ and

$$E(G_Q) := \{(i, j) \mid q_{ij} > 0\}.$$

We call the Markov chain **connected** if G_Q is strongly connected, that is, for all $i, j \in [n]$ there is a path from i to j in G_Q .

Notes for Page 6.24

Remember the definition of irreducible matrices in Chapter 5 (in connection with the Perron-Frobenius Theorem) and note that a Markov chain is connected if and only if its transition matrix is irreducible.

Probability Distribution

We describe probability distributions on $[n]$ by row vectors

$\mathbf{p} = (p_1, \dots, p_n) \in \mathbb{R}^{1 \times n}$, which are non-negative with $\sum_{i=1}^n p_i = 1$. We call such vectors **probability vectors**.

We consider a Markov chain with transition matrix $Q = (q_{ij}) \in \mathbb{R}^{n \times n}$.

- ▶ We denote the initial probability distribution of the Markov chain by $\mathbf{p}_0 = (p_{01}, \dots, p_{0n})$.
If there is a single initial state i_0 , we have $\mathbf{p}_0 = \mathbf{e}_{i_0}^\top$.
- ▶ We let $\mathbf{p}_t = (p_{t1}, \dots, p_{tn})$ be the probability distribution over the state space after t steps. That is, p_{tj} is the probability that the chain is in state j after t steps. Then

$$\mathbf{p}_t = \mathbf{p}_0 Q^t. \quad (*)$$

- ▶ We define the **average probability distribution** $\mathbf{a}_t = (a_{t1}, \dots, a_{tn})$ at time t by

$$\mathbf{a}_t := \frac{1}{t} \sum_{s=1}^t \mathbf{p}_s.$$

Proof of (*).

Suppose that $\mathbf{p}_t = (p_{t1}, \dots, p_{tn})$. If the process is in state i at time t , then the probability of being in state j at time $(t+1)$ is q_{ij} . Thus the probability of being in state j at time $t+1$ is

$$p_{(t+1)j} = \sum_{i=1}^n p_{ti} q_{ij} = (\mathbf{p}_t Q)_j.$$

This proves that $\mathbf{p}_{t+1} = \mathbf{p}_t Q$. Now (*) follows by induction. □

Fundamental Theorem of Markov Chains

Theorem 6.9

Let $Q \in \mathbb{R}^{n \times n}$ be the transition matrix of a connected Markov chain. Then there is a unique probability vector $\boldsymbol{\pi} \in \mathbb{R}^{1 \times n}$ such that $\boldsymbol{\pi}Q = \boldsymbol{\pi}$. Moreover, for every initial distribution $\mathbf{p}_0 \in \mathbb{R}^{1 \times n}$ the average probability distributions \mathbf{a}_t satisfy

$$\lim_{t \rightarrow \infty} \mathbf{a}_t = \boldsymbol{\pi}.$$

Recall that the **spectral radius** $\rho(A)$ of a square matrix A is the maximum absolute value of an eigenvalue of A .

Lemma 6.10

Let $Q \in \mathbb{R}^{n \times n}$ be the transition matrix of a connected Markov chain. Then the spectral radius of Q is 1.

Proof.

If $n = 1$ then $Q = (1)$, then the assertion is trivial. Assume $n \geq 2$.

By the Perron-Frobenius Theorem (Theorem 5.21), $\rho := \rho(Q) > 0$ is a left eigenvalue of Q . Let $\mathbf{v} = (v_1, \dots, v_n)^\top$ be the left Perron vector of Q . Then $v_i > 0$ for all i and $\mathbf{v}^\top Q = \rho \mathbf{v}^\top$. Thus

$$\rho \sum_{i=1}^n v_i = \sum_{j=1}^n (\mathbf{v}^\top Q)_j = \sum_{j=1}^n \sum_{i=1}^n v_i q_{ij} = \sum_{i=1}^n v_i \sum_{j=1}^n q_{ij} = \sum_{i=1}^n v_i.$$

As $v_i > 0$ for all i , this implies $\rho = 1$.

□

Proof of the theorem.

Again, without loss of generality we assume $n \geq 2$. Let $\mathbf{v} = (v_1, \dots, v_n)^\top$ be the left Perron vector of Q . Then $v_i > 0$ for all i and $\mathbf{v}^\top Q = \mathbf{v}^\top$. Let $z := \sum_{i=1}^n v_i$ and

$$\boldsymbol{\pi} := \frac{\mathbf{v}^\top}{z}.$$

Then $\boldsymbol{\pi} = (\pi_1, \dots, \pi_n) \in \mathbb{R}^{1 \times n}$ is a probability vector with $\boldsymbol{\pi} Q = \boldsymbol{\pi}$. The uniqueness follows from the uniqueness of the left Perron vector.

Observe that the right Perron vector of Q is $\mathbf{u} := \frac{1}{\sqrt{n}} \mathbf{1}$. By the previous lemma we have $\rho(Q) = 1$. Thus by the Limit Theorem for Nonnegative Matrices (Theorem 5.23), it holds that

$$\lim_{t \rightarrow \infty} \frac{1}{t} \sum_{s=1}^t Q^s = \frac{1}{\langle \mathbf{u}, \mathbf{v} \rangle} \mathbf{u} \cdot \mathbf{v}^\top = \frac{\sqrt{n}}{z \langle \mathbf{1}, \boldsymbol{\pi}^\top \rangle} \frac{z}{\sqrt{n}} \mathbf{1} \cdot \boldsymbol{\pi} = \mathbf{1} \cdot \boldsymbol{\pi}.$$

Here we use $\mathbf{u} = \frac{1}{\sqrt{n}} \mathbf{1}$, $\mathbf{v}^\top = z \boldsymbol{\pi}$, and $\langle \mathbf{1}, \boldsymbol{\pi}^\top \rangle = \sum_{i=1}^n \pi_i = 1$.

Thus with initial distribution $\mathbf{p}_0 = (p_{01}, \dots, p_{0n})$,

$$\lim_{t \rightarrow \infty} \mathbf{a}_t = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{s=1}^t \mathbf{p}_0 Q^s = \mathbf{p}_0 \cdot \mathbf{1} \cdot \boldsymbol{\pi} = \left(\sum_{i=1}^n p_{0i} \right) \boldsymbol{\pi} = \boldsymbol{\pi}.$$

□

Recognising the Stationary Distribution

Lemma 6.11

Let $Q = (q_{ij}) \in \mathbb{R}^{n \times n}$ be the transition matrix of a connected Markov chain, and let $\boldsymbol{\pi} = (\pi_1, \dots, \pi_n) \in \mathbb{R}^{1 \times n}$ be a probability vector satisfying

$$\pi_i q_{ij} = \pi_j q_{ji}$$

for all $i, j \in [n]$. Then $\boldsymbol{\pi}$ is the stationary distribution of the Markov chain.

Proof.

Summing both sides of the equation $\pi_i q_{ij} = \pi_j q_{ji}$, we obtain

$$\pi_i = \sum_{j=1}^n \pi_j q_{ji} = (\boldsymbol{\pi} Q)_i$$

Thus $\boldsymbol{\pi} = \boldsymbol{\pi} Q$, and $\boldsymbol{\pi}$ is the stationary distribution. □

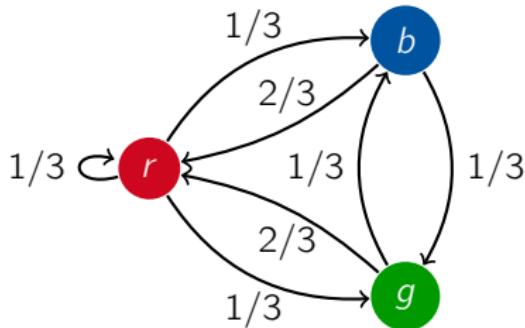
Remark.

Note that, if there is probability vector that fulfils the above condition it is the stationary distribution. However, a stationary distribution may exist and not be of the above form.

The above so-called **detailed balance condition** assumes the Markov chain is **reversible**, i.e., the probability flow from state i to j is the same as from j to i . However, in a general connected graph there may be no such bidirectional flow, despite the chain being stationary.

Example 6.12

Consider the Markov chain described by the following graph (cf. Example 6.8):



The transition matrix of this Markov chain is

$$Q = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{2}{3} & 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & 0 \end{pmatrix}.$$

Example 6.12 (cont'd)

With the initial distribution $\mathbf{p}_0 = (1/3, 1/3, 1/3)$, the distributions \mathbf{p}_t and \mathbf{a}_t evolve as follows:

t	\mathbf{p}_t	\mathbf{a}_t
0	(0.33, 0.33, 0.33)	
1	(0.56, 0.22, 0.22)	(0.56, 0.22, 0.22)
2	(0.48, 0.26, 0.26)	(0.52, 0.24, 0.24)
3	(0.51, 0.25, 0.25)	(0.51, 0.24, 0.24)
4	(0.50, 0.25, 0.25)	(0.51, 0.24, 0.24)
5	(0.50, 0.25, 0.25)	(0.51, 0.25, 0.25)
6	(0.50, 0.25, 0.25)	(0.51, 0.25, 0.25)
7	(0.50, 0.25, 0.25)	(0.51, 0.25, 0.25)
8	(0.50, 0.25, 0.25)	(0.51, 0.25, 0.25)
9	(0.50, 0.25, 0.25)	(0.50, 0.25, 0.25)
10	(0.50, 0.25, 0.25)	(0.50, 0.25, 0.25)

Example 6.12 (cont'd)

With the initial distribution $\mathbf{p}_0 = (0, 0, 1)$, the distributions \mathbf{p}_t and \mathbf{a}_t evolve as follows:

t	\mathbf{p}_t	\mathbf{a}_t
0	(0.00, 0.00, 1.00)	
1	(0.67, 0.33, 0.00)	(0.67, 0.33, 0.00)
2	(0.44, 0.22, 0.33)	(0.56, 0.28, 0.17)
3	(0.52, 0.26, 0.22)	(0.54, 0.27, 0.19)
4	(0.49, 0.25, 0.26)	(0.53, 0.27, 0.20)
5	(0.50, 0.25, 0.25)	(0.53, 0.26, 0.21)
6	(0.50, 0.25, 0.25)	(0.52, 0.26, 0.22)
7	(0.50, 0.25, 0.25)	(0.52, 0.26, 0.22)
8	(0.50, 0.25, 0.25)	(0.52, 0.26, 0.23)
9	(0.50, 0.25, 0.25)	(0.51, 0.26, 0.23)
10	(0.50, 0.25, 0.25)	(0.51, 0.26, 0.23)
:	:	:
38	(0.50, 0.25, 0.25)	(0.50, 0.25, 0.25)

Example 6.12 (cont'd)

Recall the transition matrix:

$$Q = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{2}{3} & 0 & \frac{1}{3} \\ \frac{2}{3} & \frac{1}{3} & 0 \end{pmatrix}.$$

We conjecture that $\boldsymbol{\pi} := (1/2, 1/4, 1/4)$ is the stationary distribution.
Indeed, we have

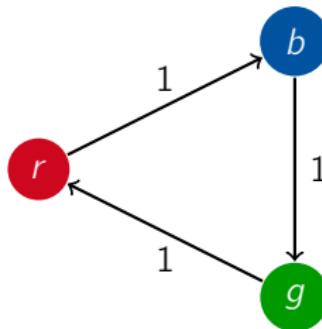
$$\boldsymbol{\pi} Q = \left(\frac{1}{6} + \frac{1}{6} + \frac{1}{6}, \frac{1}{6} + \frac{1}{12}, \frac{1}{6} + \frac{1}{12} \right) = \boldsymbol{\pi}.$$

Or we can check with Lemma 6.11 that $\pi_i q_{ij} = \pi_j q_{ji}$ for all $i, j \in [n]$.
For example,

$$\pi_1 q_{13} = \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6} = \frac{1}{4} \cdot \frac{2}{3} = \pi_3 q_{31}.$$

Example 6.13

Consider the Markov chain described by the following graph:



The transition matrix of this Markov chain is

$$Q = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

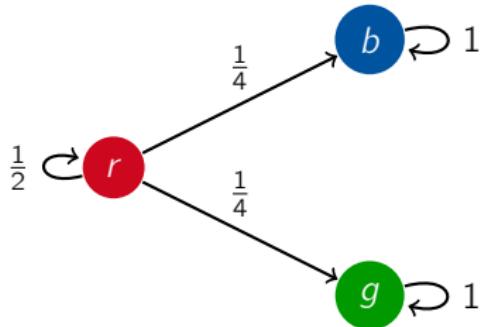
Example 6.13 (cont'd)

With the initial distribution $\mathbf{p}_0 = (1, 0, 0)$, the distributions \mathbf{p}_t and \mathbf{a}_t evolve as follows:

t	\mathbf{p}_t	\mathbf{a}_t
0	(1.00, 0.00, 0.00)	
1	(0.00, 1.00, 0.00)	(0.00, 1.00, 0.00)
2	(0.00, 0.00, 1.00)	(0.00, 0.50, 0.50)
3	(1.00, 0.00, 0.00)	(0.33, 0.33, 0.33)
4	(0.00, 1.00, 0.00)	(0.25, 0.50, 0.25)
5	(0.00, 0.00, 1.00)	(0.20, 0.40, 0.40)
6	(1.00, 0.00, 0.00)	(0.33, 0.33, 0.33)
7	(0.00, 1.00, 0.00)	(0.29, 0.43, 0.29)
8	(0.00, 0.00, 1.00)	(0.25, 0.38, 0.38)
9	(1.00, 0.00, 0.00)	(0.33, 0.33, 0.33)
10	(0.00, 1.00, 0.00)	(0.30, 0.40, 0.30)
:	:	:
198	(1.00, 0.00, 0.00)	(0.33, 0.33, 0.33)
199	(0.00, 1.00, 0.00)	(0.33, 0.34, 0.33)
200	(0.00, 0.00, 1.00)	(0.33, 0.34, 0.34)

Example 6.14

Consider the Markov chain described by the following graph:



Note that this Markov chain is **not connected!**

The transition matrix of this Markov chain is

$$Q = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Example 6.14 (cont'd)

With the initial distribution $\mathbf{p}_0 = (0, 1, 0)$, the distributions \mathbf{p}_t and \mathbf{a}_t evolve as follows:

t	\mathbf{p}_t	\mathbf{a}_t
0	(0, 1, 0)	
1	(0, 1, 0)	(0, 1, 0)
2	(0, 1, 0)	(0, 1, 0)
3	(0, 1, 0)	(0, 1, 0)

With the initial distribution $\mathbf{p}_0 = (0, 0, 1)$, the distributions \mathbf{p}_t and \mathbf{a}_t evolve as follows:

t	\mathbf{p}_t	\mathbf{a}_t
0	(0, 0, 1)	
1	(0, 0, 1)	(0, 0, 1)
2	(0, 0, 1)	(0, 0, 1)
3	(0, 0, 1)	(0, 0, 1)

There is no unique stationary distribution.

Aperiodic and Ergodic Markov Chains

A Markov chain \mathcal{Q} with graph $G_{\mathcal{Q}}$ is **aperiodic** if the greatest common divisor of the lengths of all cycles in $G_{\mathcal{Q}}$ is 1.

A Markov chain is **ergodic** if it is connected and aperiodic.

Theorem 6.15

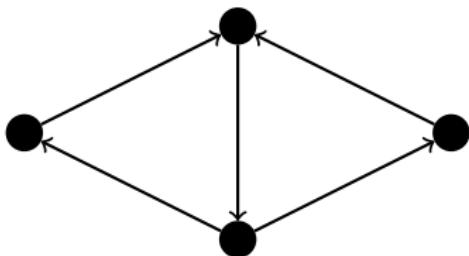
Let \mathcal{Q} be an ergodic Markov chain. Let $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \dots$ be the sequence of probability distributions reached by \mathcal{Q} from an arbitrary initial distribution \mathbf{p}_0 , and let $\boldsymbol{\pi}$ be the stationary distribution.

Then

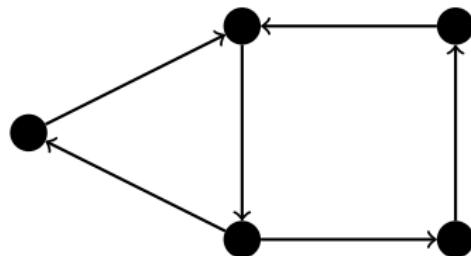
$$\lim_{t \rightarrow \infty} \mathbf{p}_t = \boldsymbol{\pi}.$$

(Proof omitted.)

Notes for Page 6.36



connected, but periodic



ergodic

Making a Markov Chain Ergodic

Lemma 6.16

Let $Q \in \mathbb{R}^{n \times n}$ be the transition matrix of a connected Markov chain \mathcal{Q} . Then for every α with $0 < \alpha < 1$, the matrix

$$\alpha Q + (1 - \alpha)I,$$

where I is the $(n \times n)$ identity matrix, is the transition matrix of an ergodic Markov chain with the same stationary distribution as \mathcal{Q} .

Proof.

Let $Q_\alpha := \alpha Q + (1 - \alpha)I$ and $G_\alpha := G_{Q_\alpha}$.

Then G_α is the graph obtained from G_Q adding a loop at every vertex. As $G_Q \subseteq G_\alpha$ and G_Q is strongly connected, G_α is strongly connected. Q_α is aperiodic, because G_α has cycles of length 1. Thus the Markov chain \mathcal{Q}_α is ergodic.

Let π be the stationary distribution of \mathcal{Q} . Then

$$\pi Q_\alpha = \alpha \pi Q + (1 - \alpha) \pi I = \alpha \pi + (1 - \alpha) \pi = \pi.$$

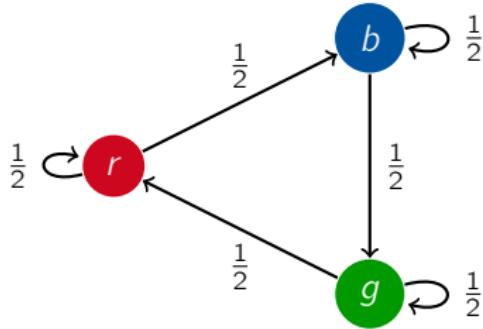
Thus π is also the stationary distribution of \mathcal{Q}_α . □

Example 6.17

Applying the operation of Lemma 6.16 with $\alpha = 1/2$ to the Markov chain of Example 6.13, we obtain the Markov chain with transition matrix

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}$$

and graph



Example 6.17 (cont'd)

With the initial distribution $\mathbf{p}_0 = (1, 0, 0)$, the distribution \mathbf{p}_t evolves as follows:

t	\mathbf{p}_t
0	(1, 0, 0)
1	(0.50, 0.50, 0)
2	(0.25, 0.50, 0.25)
3	(0.25, 0.38, 0.38)
4	(0.31, 0.31, 0.38)
5	(0.34, 0.31, 0.34)
6	(0.34, 0.33, 0.33)
7	(0.34, 0.34, 0.33)
8	(0.33, 0.34, 0.33)
9	(0.33, 0.33, 0.33)

The Stationary Distribution as Eigenvector

Let Q be the transition matrix of a connected Markov chain, and let π be its stationary distribution.

- ▶ π^\top is an eigenvector of Q^\top (or equivalently, π is a left eigenvector of Q) with eigenvalue 1.
- ▶ Iterating the Markov chain to compute the stationary distribution is essentially just the power iteration method for computing an eigenvector corresponding to the largest eigenvalue.
- ▶ If the Markov chain is ergodic, the method is guaranteed to converge; otherwise we need to average.

6.3 Markov Chain Monte Carlo

Monte Carlo

We want to estimate the expected value $\mu = E(f)$ of some random variable f defined on a probability space (Ω, \mathcal{P}) by taking samples $\omega_1, \dots, \omega_m$ and then returning $\hat{\mu} := \frac{1}{m} \sum_{i=1}^m f(\omega_i)$ as an estimator.
The difficulty is to sample from the (large) probability space (Ω, \mathcal{P}) .

Markov Chain Monte Carlo (MCMC)

- ▶ To sample from (Ω, \mathcal{P}) , we set up an ergodic Markov chain with state space Ω and stationary distribution \mathcal{P} .
- ▶ We simulate the Markov chain until the distribution is close to the stationary distribution.
- ▶ Then we can take the current state as a sample from (Ω, \mathcal{P}) .
- ▶ As for rejection sampling, it will usually be enough to know \mathcal{P} up to a normalising constant, that is, have access to $\mathcal{D} := Z\mathcal{P}$.

Let \mathcal{Q} be a Markov chain with state space Ω . We usually assume \mathcal{Q} to be ergodic.

- ▶ We denote the transition probabilities of \mathcal{Q} by $\mathcal{Q}(\omega, \omega')$, for $\omega, \omega' \in \Omega$.
- ▶ For every probability distribution \mathcal{P}_0 on Ω we denote the distribution of \mathcal{Q} after t steps with initial distribution \mathcal{P}_0 by $\mathcal{Q}_{\mathcal{P}_0, t}$.
- ▶ If \mathcal{P}_0 is a 1-point distribution with $\mathcal{P}_0(\omega_0) = 1$ for some $\omega_0 \in \Omega$, we write $\mathcal{Q}_{\omega_0, t}$ instead of $\mathcal{Q}_{\mathcal{P}_0, t}$. This is the distribution we obtain if we run \mathcal{Q} for t steps with initial state ω_0 .
- ▶ We denote the stationary distribution of \mathcal{Q} by \mathcal{Q}_∞ .

Notes for Page 6.43

Our notation for Markov chains so far emphasised the connections with linear algebra and was based on matrices and vectors. From now on, it will be convenient to use a more “probability based” notation.

Note that for all $t \in \mathbb{N}$ and all $\omega_t \in \Omega$ we have

$$Q_{P_0,t}(\omega_t) = \sum_{\omega_0, \dots, \omega_{t-1} \in \Omega} P_0(\omega_0) \prod_{i=1}^t Q(\omega_{i-1}, \omega_i).$$

Total Variation Distance

The **total variation distance** between two probability distributions $\mathcal{P}, \mathcal{P}'$ over the same state space Ω is

$$\text{dist}_{\text{TV}}(\mathcal{P}, \mathcal{P}') := \max_{A \subseteq \Omega} |\mathcal{P}(A) - \mathcal{P}'(A)|$$

Lemma 6.18

$$\text{dist}_{\text{TV}}(\mathcal{P}, \mathcal{P}') = \frac{1}{2} \sum_{\omega \in \Omega} |\mathcal{P}(\omega) - \mathcal{P}'(\omega)|.$$

That is, the total variation distance is one half of the ℓ_1 -norm between the two probability distributions if we view them as vectors in $[0, 1]^{\Omega}$.

Proof of Lemma 6.18.

Let Ω^+ be the set of all $\omega \in \Omega$ with $\mathcal{P}(\omega) \geq \mathcal{P}'(\omega)$ and $\Omega^- := \Omega \setminus \Omega^+$. Then

$$\mathcal{P}(\Omega^+) + \mathcal{P}(\Omega^-) = \mathcal{P}'(\Omega^+) + \mathcal{P}'(\Omega^-) = 1.$$

This implies

$$\mathcal{P}(\Omega^+) - \mathcal{P}'(\Omega^+) = \mathcal{P}'(\Omega^-) - \mathcal{P}(\Omega^-).$$

Observe that

$$\max \{ \mathcal{P}(A) - \mathcal{P}'(A) \mid A \subseteq \Omega \} = \mathcal{P}(\Omega^+) - \mathcal{P}'(\Omega^+)$$

and

$$\max \{ \mathcal{P}'(A) - \mathcal{P}(A) \mid A \subseteq \Omega \} = \mathcal{P}'(\Omega^-) - \mathcal{P}(\Omega^-).$$

Thus

$$\text{dist}_{\text{TV}}(\mathcal{P}, \mathcal{P}') = \mathcal{P}(\Omega^+) - \mathcal{P}'(\Omega^+) = \mathcal{P}'(\Omega^-) - \mathcal{P}(\Omega^-).$$

It follows that

$$2 \text{dist}_{\text{TV}}(\mathcal{P}, \mathcal{P}') = \mathcal{P}(\Omega^+) - \mathcal{P}'(\Omega^+) + \mathcal{P}'(\Omega^-) - \mathcal{P}(\Omega^-)$$

$$\begin{aligned}
&= \sum_{\omega \in \Omega^+} \underbrace{(\mathcal{P}(\omega) - \mathcal{P}'(\omega))}_{\geq 0} + \sum_{\omega \in \Omega^-} \underbrace{(\mathcal{P}'(\omega) - \mathcal{P}(\omega))}_{\geq 0} \\
&= \sum_{\omega \in \Omega} |\mathcal{P}(\omega) - \mathcal{P}'(\omega)|.
\end{aligned}$$

□

Let \mathcal{Q} be an ergodic Markov chain with (finite) state space Ω and stationary distribution \mathcal{P} .

By Theorem 6.15, for all $\omega_0, \omega \in \Omega$ we have $\lim_{t \rightarrow \infty} \mathcal{Q}_{\omega_0, t}(\omega) = \mathcal{P}(\omega)$.
Thus

$$\lim_{t \rightarrow \infty} \text{dist}_{\text{TV}}(\mathcal{Q}_{\omega_0, t}, \mathcal{P}) = 0.$$

Definition 6.19

The **mixing time** of \mathcal{Q} is the function $T_{\mathcal{Q}} : (0, 1) \rightarrow \mathbb{N}$ defined by

$$T_{\mathcal{Q}}(\varepsilon) = \min_{t \in \mathbb{N}} \left\{ t \in \mathbb{N} \mid \forall \omega_0 \in \Omega : \text{dist}_{\text{TV}}(\mathcal{Q}_{\omega_0, t}, \mathcal{P}) \leq \varepsilon \right\}.$$

Approximation with a Markov Chain

Let (Ω, \mathcal{P}) be a (finite) probability space and f a random variable on (Ω, \mathcal{P}) . Assume that $|f(\omega)| \leq b$ for all $\omega \in \Omega$. We want to estimate

$$\mu := E(f).$$

Let \mathcal{Q} be an ergodic Markov chain with state space Ω and stationary distribution \mathcal{P} .

Lemma 6.20

For all $t \in \mathbb{N}$ and all $\omega_0 \in \Omega$, we have

$$\left| \sum_{\omega \in \Omega} f(\omega) \mathcal{Q}_{\omega_0, t}(\omega) - \mu \right| \leq 2b \cdot \text{dist}_{\text{TV}}(\mathcal{Q}_{\omega_0, t}, \mathcal{P}).$$

Notes for Page 6.46

Proof of Lemma 6.20.

$$\begin{aligned} \left| \sum_{\omega \in \Omega} f(\omega) Q_{\omega_0, t}(\omega) - \mu \right| &= \left| \sum_{\omega \in \Omega} f(\omega) Q_{\omega_0, t}(\omega) - \sum_{\omega \in \Omega} f(\omega) P(\omega) \right| \\ &\leq \sum_{\omega \in \Omega} |f(\omega)| \cdot |Q_{\omega_0, t}(\omega) - P(\omega)| \\ &\leq 2b \cdot \frac{1}{2} \sum_{\omega \in \Omega} |Q_{\omega_0, t}(\omega) - P(\omega)| \\ &= 2b \cdot \text{dist}_{\text{TV}}(Q_{\omega_0, t}, P) \quad \text{by Lemma 6.18.} \end{aligned}$$

□

MCMC Theorem

Theorem 6.21 (MCMC Theorem)

Let (Ω, \mathcal{P}) be a (finite) probability space. Let f a random variable on (Ω, \mathcal{P}) such that $|f(\omega)| \leq b$ for all $\omega \in \Omega$, and let $\mu := E(f)$. Let \mathcal{Q} be an ergodic Markov chain with state space Ω and stationary distribution \mathcal{P} .

Let $\varepsilon, \delta > 0$ and $m, t \in \mathbb{N}$ such that $m \geq \frac{4b}{\varepsilon^2} \ln\left(\frac{2}{\delta}\right)$ and $t \geq T_{\mathcal{Q}}\left(\frac{\varepsilon}{4b}\right)$. Let $\omega_0 \in \Omega$, and for each $i \in [m]$, let ω_i be the state reached in an independent run of \mathcal{Q} after t steps. Let

$$\hat{\mu} := \frac{1}{m} \sum_{i=1}^m f(\omega_i).$$

Then

$$\Pr(|\hat{\mu} - \mu| \leq \varepsilon) \geq 1 - \delta.$$

Notes for Page 6.47

Proof of the MCMC Theorem.

For $i \in [m]$, let $X_i := f(\omega_i)$. Then we can view the X_i as independent random variables on the space $(\Omega, \mathcal{Q}_{\omega_0, t})$. Note that $\text{rg}(X_i) \in [-b, b]$ and that

$$\mathbb{E}(X_i) = \sum_{\omega \in \Omega} f(\omega) \mathcal{Q}_{\omega_0, t}(\omega)$$

Let $X = \sum_{i=1} X_i$ and note that $\hat{\mu} = \frac{1}{m} X$. We have

$$\begin{aligned} \Pr \left(\left| \hat{\mu} - \frac{\mathbb{E}(X)}{m} \right| \geq \frac{\varepsilon}{2} \right) &= \Pr \left(|X - \mathbb{E}(X)| \geq \frac{\varepsilon}{2} m \right) \\ &\leq 2 \exp \left(-\frac{\varepsilon^2}{4b} m \right) && \text{by Lemma 6.2 (Hoeffding)} \\ &\leq \delta && \text{because } m \geq \frac{4b}{\varepsilon^2} \ln \left(\frac{2}{\delta} \right). \end{aligned}$$

By Lemma 6.20, we have

$$\left| \frac{\mathbb{E}(X)}{m} - \mu \right| = \left| \sum_{\omega \in \Omega} f(\omega) \mathcal{Q}_{\omega_0, t}(\omega) - \mu \right| \leq 2b \text{dist}_{\text{TV}}(\mathcal{Q}_{\omega_0, t}, \mathcal{P}) \leq \frac{\varepsilon}{2},$$

because $t \geq T_Q\left(\frac{\varepsilon}{4b}\right)$. Thus

$$\begin{aligned}\Pr(|\hat{\mu} - \mu| \geq \varepsilon) &\leq \Pr\left(\left|\hat{\mu} - \frac{\mathbb{E}(X)}{m}\right| + \left|\frac{\mathbb{E}(X)}{m} - \mu\right| \geq \varepsilon\right) \\ &\leq \Pr\left(\left|\hat{\mu} - \frac{\mathbb{E}(X)}{m}\right| \geq \frac{\varepsilon}{2}\right) \\ &\leq \delta.\end{aligned}$$



Metropolis-Hastings Sampling

We want to sample from the probability space (Ω, \mathcal{P}) , where $\mathcal{P} = \frac{\mathcal{D}}{Z}$ for an unknown normalising constant Z .

- ▶ Our Markov chain will be based on a connected undirected graph \mathcal{G} of maximum degree Δ with vertex set $V(\mathcal{G}) = \Omega$.
- ▶ Let \mathcal{Q} be the Markov chain with state space Ω and transition probabilities:

$$\mathcal{Q}(\omega, \omega') := \begin{cases} \frac{1}{\Delta+1} & \text{if } \omega\omega' \in E(\mathcal{G}) \text{ and } \mathcal{D}(\omega') \geq \mathcal{D}(\omega), \\ \frac{1}{\Delta+1} \cdot \frac{\mathcal{D}(\omega')}{\mathcal{D}(\omega)} & \text{if } \omega\omega' \in E(\mathcal{G}) \text{ and } \mathcal{D}(\omega') < \mathcal{D}(\omega), \\ 1 - \sum_{\omega'' \in N_{\mathcal{G}}(\omega)} q_{\omega\omega''} & \text{if } \omega = \omega' \\ 0 & \text{otherwise} \end{cases}$$

Example

We may have $\Omega = \{0, 1\}^d$. We define the graph \mathcal{G} by adding edges between all $\mathbf{x}, \mathbf{y} \in \Omega$ that differ in exactly one coordinate.

Transition Algorithm

The transition probabilities result from the following algorithm for choosing the next state ω' from the current state ω :

1. $b \leftarrow \begin{cases} 1 & \text{with probability } \frac{|N(\omega)|}{\Delta+1}, \\ 0 & \text{otherwise} \end{cases}$
2. if $b = 1$ then
3. choose a neighbour $\omega'' \in N_G(\omega)$ uniformly at random
4. if $\mathcal{D}(\omega'') \geq \mathcal{D}(\omega)$ then
5. $\omega' \leftarrow \omega''$
6. else
7. $\omega' \leftarrow \begin{cases} \omega'' \text{ with probability } \mathcal{D}(\omega'')/\mathcal{D}(\omega) \\ \omega \text{ with probability } 1 - \mathcal{D}(\omega'')/\mathcal{D}(\omega) \end{cases}$
8. else
9. $\omega' \leftarrow \omega$
10. return ω'

Stationary Distribution

Theorem 6.22

\mathcal{Q} is an ergodic Markov chain with stationary distribution \mathcal{P} .

Proof.

To see that \mathcal{Q} is ergodic, just note that it is connected, because the graph \mathcal{G} is connected, and that $\mathcal{Q}(\omega, \omega) \geq \frac{1}{\Delta+1} > 0$ for all $\omega \in \Omega$.

To see that the stationary distribution is \mathcal{P} , we shall prove that

$$\mathcal{D}(\omega)\mathcal{Q}(\omega, \omega') = \mathcal{D}(\omega')\mathcal{Q}(\omega', \omega). \quad (*)$$

Since $\mathcal{P}(\omega) = \mathcal{D}(\omega)/Z$, this implies $\mathcal{P}(\omega)\mathcal{Q}(\omega, \omega') = \mathcal{P}(\omega')\mathcal{Q}(\omega', \omega)$, and it follows from Lemma 6.11 that \mathcal{P} is the stationary distribution.

(*) trivially holds if $\omega = \omega'$. If $\omega \neq \omega'$ and $\omega\omega' \notin E(\mathcal{G})$, we have

$\mathcal{Q}(\omega, \omega') = \mathcal{Q}(\omega', \omega) = 0$, because the graph is undirected. Again, (*) holds.

Let us assume that $\omega\omega' \in E(\mathcal{G})$.

Then if $\mathcal{D}(\omega) = \mathcal{D}(\omega')$, we have $\mathcal{Q}(\omega, \omega') = \mathcal{Q}(\omega', \omega) = \frac{1}{\Delta+1}$ and thus $\mathcal{D}(\omega)\mathcal{Q}(\omega, \omega') = \mathcal{D}(\omega')\mathcal{Q}(\omega', \omega)$.

If $\mathcal{D}(\omega) < \mathcal{D}(\omega')$, we have $\mathcal{Q}(\omega, \omega') = \frac{1}{\Delta+1}$ and $\mathcal{Q}(\omega', \omega) = \frac{\mathcal{D}(\omega)}{(\Delta+1) \cdot \mathcal{D}(\omega')}$ and thus

$$\mathcal{D}(\omega)\mathcal{Q}(\omega, \omega') = \frac{\mathcal{D}(\omega)}{\Delta+1} = \mathcal{D}(\omega') \frac{\mathcal{D}(\omega)}{(\Delta+1) \cdot \mathcal{D}(\omega')} = \mathcal{D}(\omega')\mathcal{Q}(\omega', \omega).$$

If $\mathcal{D}(\omega) > \mathcal{D}(\omega')$, we have $\mathcal{Q}(\omega, \omega') = \frac{\mathcal{D}(\omega')}{(\Delta+1) \cdot \mathcal{D}(\omega)}$ and $\mathcal{Q}(\omega', \omega) = \frac{1}{\Delta+1}$ and thus

$$\mathcal{D}(\omega)\mathcal{Q}(\omega, \omega') = \mathcal{D}(\omega) \frac{\mathcal{D}(\omega')}{(\Delta + 1) \cdot \mathcal{D}(\omega)} = \frac{\mathcal{D}(\omega')}{\Delta + 1} = \mathcal{D}(\omega')\mathcal{Q}(\omega', \omega).$$



Example: Sampling Matchings

Let G be a graph, and let $\mathcal{M} \subseteq 2^{E(G)}$ be the set of matchings of G . We will define a Markov chain with state space \mathcal{M} and the uniform distribution on \mathcal{M} as its stationary distribution.

We let \mathcal{G} be the graph with vertex set $V(\mathcal{G}) = \mathcal{M}$ and, for all $M, M' \in \mathcal{M}$, an edge between M and M' if one of the following conditions is satisfied:

- ▶ there is an $e \in M \setminus M'$ such that $M = M' \cup \{e\}$ or an $f \in M' \setminus M$ such that $M' = M \cup \{f\}$;
- ▶ there are $e \in M \setminus M', f \in M' \setminus M$ such that e and f have a common endvertex and $M \setminus \{e\} = M' \setminus \{f\}$.

Note that the maximum degree Δ of \mathcal{G} is $|E(G)|$. Furthermore, \mathcal{G} is connected because every node has a path to \emptyset .

We define \mathcal{D} by $\mathcal{D}(M) = 1$ for all $m \in \mathcal{M}$. The transition properties of the Metropolis-Hastings Markov chain with graph \mathcal{G} are

$$q_{MM'} = \begin{cases} \frac{1}{\Delta+1} & \text{if } MM' \in E(\mathcal{G}), \\ 1 - \frac{\deg_{\mathcal{G}}(M)}{\Delta+1} & \text{if } M = M', \\ 0 & \text{otherwise.} \end{cases}$$

Rapid Mixing

- ▶ It was proved in [Jerrum and Sinclair, 1989] that the mixing time of the Metropolis-Hastings Markov chain for sampling matchings is polynomially bounded in $\frac{1}{\epsilon}$ and the size of the input graph.
- ▶ This can be used to design a randomised polynomial time algorithm for estimating the number of matchings of a graph.
- ▶ Similar results also hold for perfect matchings.

6.4 Coupling of Markov Chains

Definition 6.23

Let \mathcal{Q} be a Markov chain with state space Ω . A **coupling** of \mathcal{Q} is a Markov chain \mathcal{C} with state space $\Omega \times \Omega$ such that for all $\omega_1, \omega_2, \omega'_1, \omega'_2 \in \Omega$ it holds that

$$\sum_{\omega' \in \Omega} \mathcal{C}((\omega_1, \omega_2), (\omega'_1, \omega')) = \mathcal{Q}(\omega_1, \omega'_1), \quad (\text{C1})$$

$$\sum_{\omega' \in \Omega} \mathcal{C}((\omega_1, \omega_2), (\omega', \omega'_2)) = \mathcal{Q}(\omega_2, \omega'_2). \quad (\text{C2})$$

We usually view a coupling \mathcal{C} as a pair $(\mathcal{Q}_1, \mathcal{Q}_2)$ of Markov chains running simultaneously on Ω . Then the coupling conditions (C1) and (C2) state that the marginal Markov chains \mathcal{Q}_1 and \mathcal{Q}_2 are both copies of \mathcal{Q} , but in general they are not independent.

The Coupling Lemma

Lemma 6.24

Let \mathcal{C} be a coupling of a Markov chain \mathcal{Q} with state space Ω . Suppose that for some $\varepsilon \in (0, 1)$ and $t \in \mathbb{N}$ it holds that

$$\mathcal{C}_{(\omega_{01}, \omega_{02}), t} \left(\{(\omega, \omega) \mid \omega \in \Omega\} \right) \geq 1 - \varepsilon$$

for all $\omega_{01}, \omega_{02} \in \Omega$.

Then $T_{\mathcal{Q}}(\varepsilon) \leq t$.

That is, if the two marginal chains in the coupling \mathcal{C} have coupled in the sense that with high probability they are in the same state, then they must be close to the stationary distribution.

Proof of the Coupling Lemma.

For an arbitrary ω_0 we let \mathcal{P}_0 be the probability distribution on $\Omega \times \Omega$ defined by

$$\mathcal{P}_0((\omega_1, \omega_2)) = \begin{cases} \mathcal{Q}_\infty(\omega_2) & \text{if } \omega_1 = \omega_0, \\ 0 & \text{otherwise.} \end{cases}$$

That is, \mathcal{P}_0 is the probability distribution whose first marginal distribution is the 1-point distribution on ω_0 and whose second marginal distribution is \mathcal{Q}_∞ , the stationary distribution of \mathcal{Q} . Since a Markov chain started on its stationary distribution stays in the stationary distribution, for all $A \subseteq \Omega$ and all $s \in \mathbb{N}$ we have

$$\mathcal{C}_{\mathcal{P}_0, s}(\Omega \times A) = \mathcal{Q}_{\mathcal{Q}_\infty, s}(A) = \mathcal{Q}_\infty(A).$$

Let $D = \{(\omega, \omega) \mid \omega \in \Omega\} \subseteq \Omega \times \Omega$ be the “diagonal event”, and let $\overline{D} := (\Omega \times \Omega) \setminus D$. Note that $\mathcal{C}_{\mathcal{P}_0, t}(\overline{D}) \leq \varepsilon$ by the assumption of the lemma. Let $A \in \Omega$ and $\overline{A} := \Omega \setminus A$. Then

$$\begin{aligned} \mathcal{Q}_{\omega_0, t}(A) &= \mathcal{C}_{\mathcal{P}_0, t}(A \times \Omega) \\ &\geq \mathcal{C}_{\mathcal{P}_0, t}((\Omega \times A) \cap D) \end{aligned}$$

$$\begin{aligned}
&= 1 - \mathcal{C}_{\mathcal{P}_0, t}((\Omega \times \overline{A}) \cup \overline{D}) \\
&\geq 1 - \left(\mathcal{C}_{\mathcal{P}_0, t}(\Omega \times \overline{A}) + \mathcal{C}_{\mathcal{P}_0, t}(\overline{D}) \right) \\
&\geq \mathcal{C}_{\mathcal{P}_0, t}(\Omega \times A) - \varepsilon \\
&= \mathcal{Q}_\infty(A) - \varepsilon.
\end{aligned}$$

The same argument applied to the set \overline{A} shows that $\mathcal{Q}_{\omega_0, t}(\overline{A}) \geq \mathcal{Q}_\infty(\overline{A}) - \varepsilon$, which implies $\mathcal{Q}_{\omega_0, t}(A) \leq \mathcal{Q}_\infty(A) + \varepsilon$.

Thus $\text{dist}_{\text{TV}}(\mathcal{Q}_{\omega_0, t}, \mathcal{Q}_\infty) \leq \varepsilon$. Since this holds for all $\omega_0 \in \Omega$, it follows that $T_{\mathcal{Q}}(\varepsilon) \leq t$. □

Example 1: Shuffling Cards

To shuffle a deck of n cards with values $1, \dots, n$, we repeat the following: we pick a random position $i \in [n]$ and put the card in position i on top of the deck.

Question

How long does it take till the deck is shuffled well?

Example 1 (cont'd): Markov Chain

We model our shuffling algorithm with the following Markov chain \mathcal{Q} .

- ▶ The state space Ω is S_n , the set of all permutations of $[n]$.
- ▶ For every permutation $\pi \in S_n$, we let $\pi_{i \rightarrow 1}$ be the permutation defined by

$$\pi_{i \rightarrow 1}(j) := \begin{cases} \pi(i) & \text{if } j = 1, \\ \pi(j - 1) & \text{if } 1 < j \leq i, \\ \pi(j) & \text{if } i < j \leq n. \end{cases}$$

Then the transition probabilities of \mathcal{Q} are

$$Q(\pi, \pi') := \begin{cases} \frac{1}{n} & \text{if } \pi' = \pi_{i \rightarrow 1} \text{ for some } i \in [n], \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 6.25

\mathcal{Q} is an ergodic Markov chain, and its stationary distribution is the uniform distribution on S_n .

Proof.

The chain is connected, because to reach permutation π from permutation π' , we first put $\pi(n)$ on top, then $\pi(n - 1)$, $\pi(n - 2)$, et cetera. It is aperiodic because with probability $\frac{1}{n}$, the state does not change.

Let \mathcal{P} denote the uniform distribution on S_n . To see that \mathcal{P} is the stationary distribution, we need to prove that for all $\pi \in S_n$ we have

$$\sum_{\pi' \in S_n} \mathcal{P}(\pi') Q(\pi', \pi) = \mathcal{P}(\pi),$$

that is, vector \mathcal{P} times matrix Q equals vector \mathcal{P} . Since \mathcal{P} is the uniform distribution, this amounts to proving that

$$\sum_{\pi' \in S_n} Q(\pi', \pi) = 1.$$

Since all possible transitions have probability $\frac{1}{n}$, we must prove that there are exactly n possible transitions into π . Indeed, for every i there is exactly one $\pi^{(i)}$ such that $\pi_{i \rightarrow 1}^{(i)} = \pi$, defined by

$$\pi^{(i)}(j) := \begin{cases} \pi(j+1) & \text{for } 1 \leq j < i, \\ \pi(1) & \text{for } j = i, \\ \pi(j) & \text{for } i < j \leq n. \end{cases}$$



Example 1 (cont'd): Coupling

Let \mathcal{C} be the Markov chain with state space $S_n \times S_n$ and transition probabilities

$$\mathcal{C}((\pi_1, \pi_2), (\pi'_1, \pi'_2)) = \begin{cases} \frac{1}{n} & \text{if for some } i \in [n] \text{ it holds that} \\ & \pi'_1 = (\pi_1)_{\pi_1^{-1}(i) \rightarrow 1} \text{ and} \\ & \pi'_2 = (\pi_2)_{\pi_2^{-1}(i) \rightarrow 1}, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, in each step of \mathcal{C} , we pick a random i and in both components move the card with value i to the top.

Lemma 6.26

\mathcal{C} is a coupling of \mathcal{Q} .

Proof.

For the first component, we have

$$\sum_{\pi' \in S_n} \mathcal{C}((\pi_1, \pi_2), (\pi'_1, \pi')) = \begin{cases} \frac{1}{n} & \text{if } \pi'_1 = (\pi_1)_{j \rightarrow 1} \text{ for some } j \in [n], \\ 0 & \text{otherwise,} \end{cases}$$

which is precisely $\mathcal{Q}(\pi_1, \pi'_1)$. To see this, we argue as follows.

- ▶ If $\pi'_1 \neq (\pi_1)_{j \rightarrow 1}$ for all j , the transition probability is $\mathcal{C}((\pi_1, \pi_2), (\pi'_1, \pi')) = 0$ for all π' .
- ▶ If $\pi'_1 = (\pi_1)_{j \rightarrow 1}$ for some j , then there is a unique π' such that $\mathcal{C}((\pi_1, \pi_2), (\pi'_1, \pi')) \neq 0$. In fact, $\pi' = (\pi_2)_{\pi_2^{-1}(\pi_1(j)) \rightarrow 1}$ with transition probability $\mathcal{C}((\pi_1, \pi_2), (\pi'_1, \pi')) = \frac{1}{n}$.

For the second component, the situation is symmetric. □

Example 1 (cont'd): Mixing Time

Lemma 6.27

Let $t \geq n \ln \left(\frac{n}{\varepsilon} \right)$. Then for all $\pi_{01}, \pi_{02} \in S_n$,

$$\mathcal{C}_{(\pi_{01}, \pi_{02}), t} \left(\{ (\pi, \pi) \mid \pi \in S_n \} \right) \geq 1 - \varepsilon.$$

Corollary 6.28

For all $\varepsilon \in (0, 1)$,

$$T_Q(\varepsilon) \leq n \ln \left(\frac{n}{\varepsilon} \right).$$

Proof of Lemma 6.27.

In each step the chain \mathcal{C} picks a random $i \in [n]$ and then moves from state (π_1, π_2) to $(\pi_1)_{\pi_1^{-1}(i) \rightarrow 1}, (\pi_2)_{\pi_2^{-1}(i) \rightarrow 1}$.

The key observation is that once value i has been picked the cards with value i in both components move to the same position, and they stay there in all subsequent steps. So we need to estimate the time it takes till all values have been picked.

This is the so-called *Coupon Collector* problem, and it is well known that after $n \ln \left(\frac{n}{\varepsilon} \right)$, all values have been picked with probability at least $1 - \varepsilon$. □

Example 2: Independent Sets

$G = (V, E)$ graph of order $n := |V|$ and maximum degree Δ ,
 $0 < k \leq \frac{n}{5(\Delta+1)}$

Observation 6.29

G has an independent set of size k , and we can efficiently find such an independent set by a simple greedy algorithm.

Goal

Sample independent sets of G of size k uniformly (or at least almost uniformly).

Example 2 (cont'd): Markov Chain

Let \mathcal{Q} be the following Markov chain.

- ▶ The state space Ω is the set of all independent sets of G of size k .
- ▶ For all $X \in \Omega$ and $v, w \in V$ we let

$$X_{v \rightarrow w} := \begin{cases} (X \setminus \{v\}) \cup \{w\} & \text{if } (X \setminus \{v\}) \cup \{w\} \in \Omega, \\ X & \text{otherwise.} \end{cases}$$

Intuitively, in state $X \in \Omega$, the chain picks $v \in X, w \in V$ uniformly at random and moves to $X_{v \rightarrow w}$.

Thus for all $X, Y \in \Omega$, the transition probability is

$$Q(X, Y) := \begin{cases} \frac{1}{kn} & \text{if } Y = X_{v \rightarrow w} \text{ for some } v \in X, w \in V \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 6.30

\mathcal{Q} is ergodic, and its stationary distribution is the uniform distribution on Ω .

(Proof as exercise.)

Example 2 (cont'd): Coupling

Observe that for all $X, Y \in \Omega$ we have $|X \setminus Y| = |Y \setminus X|$. Let $h_{XY} : X \rightarrow Y$ be a bijective mapping such that $h_{XY}(v) = v$ for all $v \in X \cap Y$.

Let \mathcal{C} be the Markov Chain with state space $\Omega \times \Omega$ and transition probabilities

$$\mathcal{C}((X, Y), (X', Y')) = \begin{cases} \frac{1}{kn} & \text{if for some } v \in X, w \in V \\ & \text{it holds that } X' = (X)_{v \rightarrow w} \\ & \text{and } Y' = (Y)_{h_{XY}(v) \rightarrow w}, \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 6.31

\mathcal{C} is a coupling of \mathcal{Q} .

Notes for Page 6.62

The proof is a straightforward analysis similar to the proof of Lemma 6.26 based on the observation that if we choose v uniformly at random from X then this means that $h_{XY}(v)$ is chosen uniformly at random from Y .

Example 2 (cont'd): Mixing Time

Lemma 6.32

For every $\varepsilon > 0$ there is a t that is polynomial in n and $\frac{1}{\varepsilon}$ such that for all $X_0, Y_0 \in \Omega$,

$$\mathcal{C}_{(X_0, Y_0), t} \left(\{(Z, Z) \mid Z \in \Omega\} \right) \geq 1 - \varepsilon.$$

Corollary 6.33

The mixing time $T_Q(\varepsilon)$ of Q is polynomially bounded in n and $\frac{1}{\varepsilon}$.

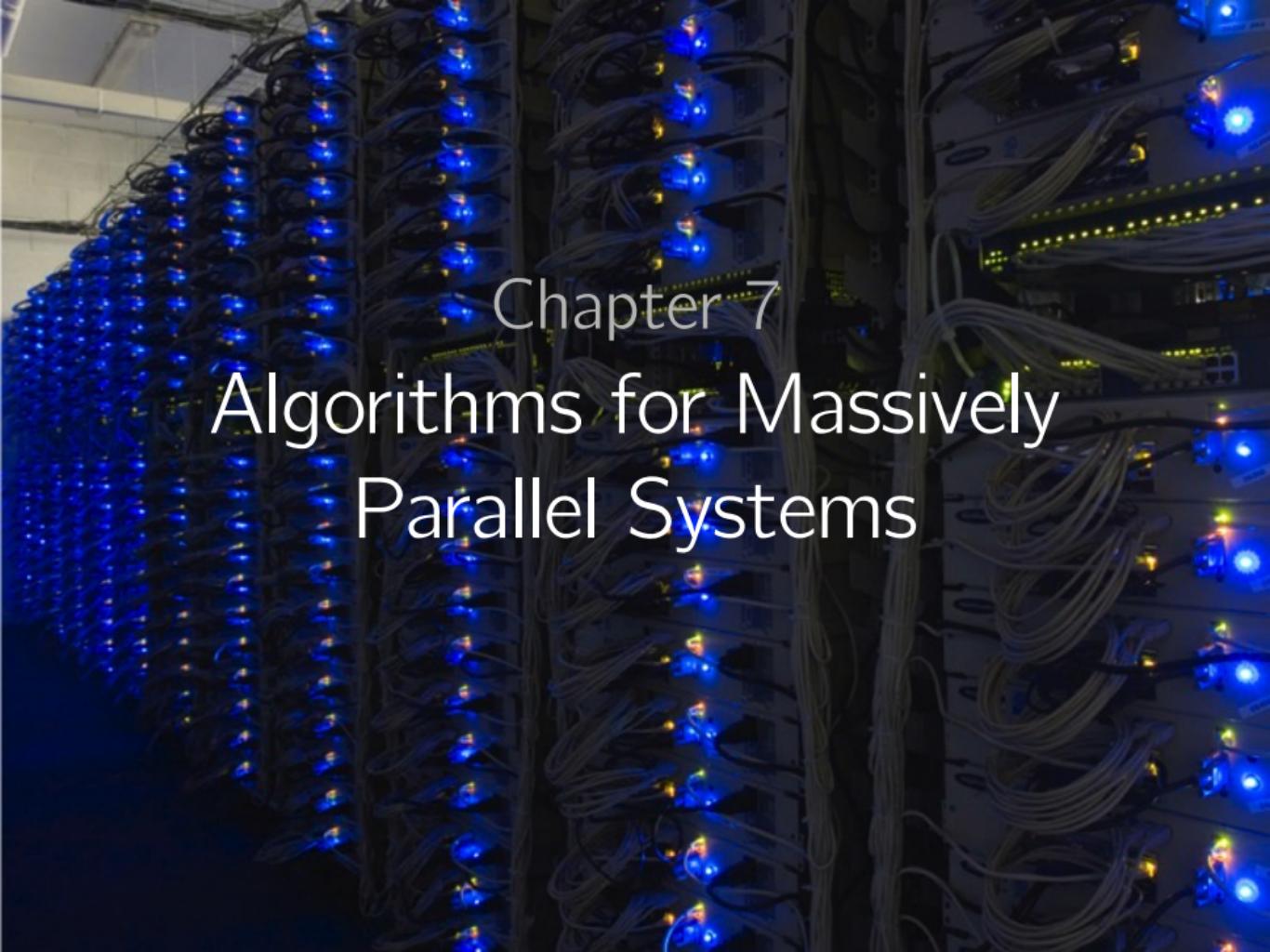
Notes for Page 6.63

The proof is based on the fact that if \mathcal{C} is in the state (X, Y) and moves to (X', Y') then it is more likely that $|X' \Delta Y'| \leq |X \Delta Y|$ than that $|X' \Delta Y'| \geq |X \Delta Y|$, and with significantly high probability we will have $|X' \Delta Y'| < |X \Delta Y|$.

To see this, observe that if the vertex w picked at random is not in $X \cup N(X) \cup Y \cup N(Y)$, then we will have $X' = (X \setminus \{v\}) \cup \{w\}$ and $Y' = (Y \setminus \{h_{XY}(v)\}) \cup \{w\}$. As $|X \cup N(X) \cup Y \cup N(Y)| < 2k(\Delta + 1) < \frac{n}{2}$, this happens with sufficiently high probability.

References

Most of the material of this chapter is covered in Chapters 7, 10, and 11 of [[Mitzenmacher and Upfal, 2005](#)]. Markov Chains and the Monte Carlo method are also covered in [[Blum et al., 2020, Chapter 4](#)]. The choice of topics there is different, but it may be interesting supplementary reading. [[MacKay, 2003, Chapter 29](#)] gives an account of Monte Carlo methods from a more applied angle.



Chapter 7

Algorithms for Massively Parallel Systems

7.1 Computing Clusters and Map-Reduce Environment

Computing Clusters

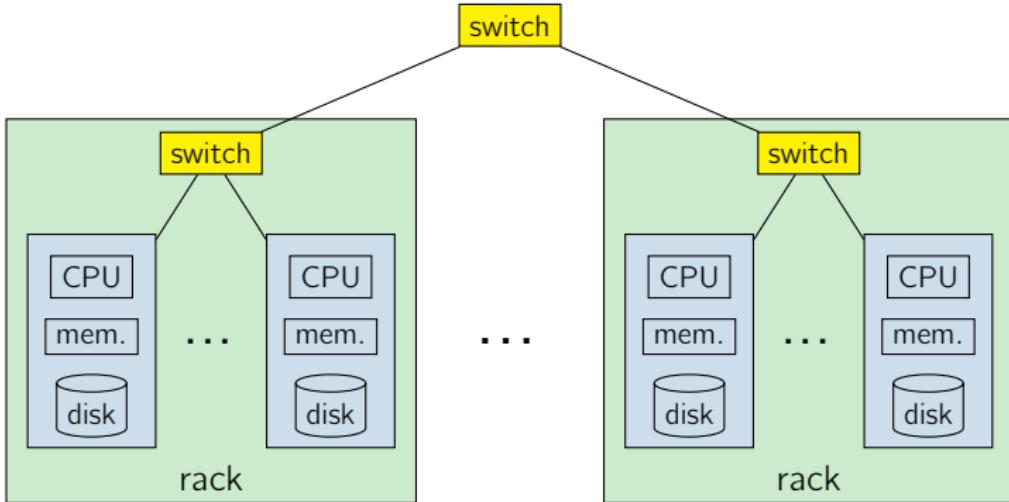
Data analysis tasks often have to be carried out on massive amounts of data. Fortunately, these analysis tasks can usually be parallelised well.

Examples

- (1) Ranking of webpages
- (2) Searches in social networks
- (3) Counting frequencies of words in collections of documents

Tasks are typically carried out on **computing clusters**, consisting of large collections of commodity hardware connected by switches, rather than special-purpose parallel computers.

Cluster Architecture



- ▶ Cluster **nodes** are standard machines with their own CPU, main memory, and disk
- ▶ Nodes operate more or less independently
- ▶ Nodes are arranged in racks of 16–64 nodes, connected by a network (typically ethernet with 1 Gbps between any pair of nodes)
- ▶ Racks are connected by another network layer (typically 2-10 Gbps between racks)

Challenges for large-scale computing on commodity hardware

- ▶ how to distribute computation
- ▶ how to write distributed programs
- ▶ how to cope with hardware failures

Node failures

- ▶ a single machine may run for about 1000 days
- ▶ if you have a cluster of 1000 nodes, you may expect one node failure every day

Distributed File Systems (DFS)

Principles

- ▶ bring data close to computation, i.e., store on local disks
- ▶ store data multiple times for reliability (typically 3 times)

Typical usage pattern

- ▶ huge files (in the terabyte range)
- ▶ frequent reads and appends, but rare in-place updates

Examples

Google's GFS and Hadoop's HDFS

Distributed File Systems (cont'd)

Data chunks

- ▶ file is split into **chunks** (typically 64 MB)
- ▶ each chunk is replicated (typically 3 times)
- ▶ replicas stored in different racks (if possible)

Master node

- ▶ stores metadata about where file chunks are stored
- ▶ might be replicated as well

Map-Reduce Programming Model

Map-Reduce is a programming model for carrying out data analysis tasks on computing clusters with input files stored in DFS.

Computation proceeds in 3 phases:

Map

- ▶ Extract information from input file and emit key-value pairs.
- ▶ Carried out in parallel on different nodes; each **map task** gets a chunk or several chunks of the input file

Shuffle

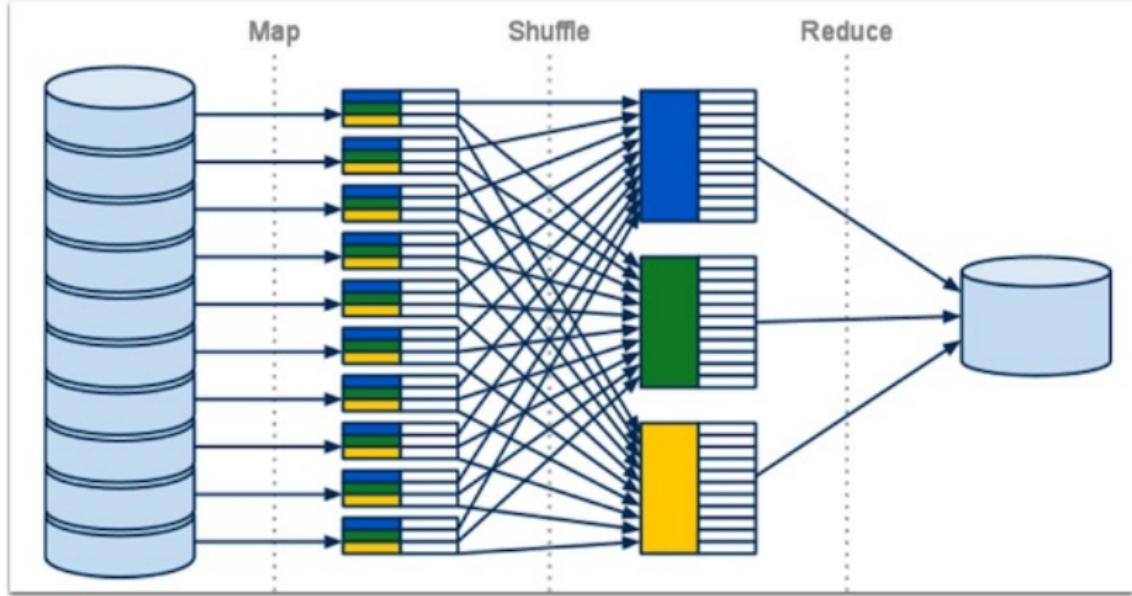
- ▶ Group key-value pairs emitted in map phase by key.

Reduce

- ▶ Combine (aggregate, summarise, filter, etc.) groups of key-value pairs with same key and emit output.
- ▶ Carried out in parallel on different nodes; each **reduce task** gets one or several keys with associated value lists.

The user has to provide a **MAP** and a **REDUCE** function, the system takes care of the rest.

Schematic View on Map-Reduce



(from *Artificial Intelligence in Motion* blog)

- ▶ Formally, the input and output are also required to consist of key-value pairs. This allows the composition of several Map-Reduce processes.
- ▶ The user-provided **MAP** function takes as input a single key-value pair and emits a list of zero or more key value pairs.
Each map task applies the **MAP** function to all key-value pairs in its chunk(s) of the input file.
- ▶ The user-provided **REDUCE** function takes as input a single key and a list of associated values and emits a list of zero or more key value pairs.
Each reduce task applies the **REDUCE** function to all its keys.

Example: Counting Words

Task

Count word frequencies in a collection of text documents.

Input: Key-value pairs (document name, text of document).

Output: Key-value pairs (word, count).

Map function

1. `function MAP(key, value)`
2. `for all words w in $value$ do`
3. `emit ($w, 1$)`

Reduce function

1. `function REDUCE(key, values)`
2. `count $\leftarrow 0$`
3. `for all v in $values$ do`
4. `count $\leftarrow count + v$`
5. `emit (key, count)`

Map-Reduce environment

Takes care of:

- ▶ partitioning the input data into map tasks and partitioning the intermediate data into reduce tasks
- ▶ assigning tasks to nodes (called **map workers** and **reduce workers**) and scheduling execution
- ▶ executing the shuffle phase
- ▶ handling communication between nodes
- ▶ dealing with node failures

- ▶ Input and final output are stored in DFS
- ▶ Scheduler tries to schedule map tasks close to physical storage location of input data (same node or at least same rack)
- ▶ Intermediate results are stored in local file system at map and reduce workers
- ▶ Output is often input to another Map-Reduce process

Master Node

- ▶ creates and schedules map tasks and reduce tasks
- ▶ maintains task status (idle, in-progress, completed) for each map and each reduce task
- ▶ pings workers periodically to detect failures
- ▶ maintains information on location of intermediate results on local disks

Coping with Node Failures

Map worker failure

- ▶ map tasks completed or in-progress at worker are reset to idle
- ▶ map tasks are rescheduled to another worker

Reduce worker failure

- ▶ reduce tasks in progress at worker are reset to idle and assigned to another worker
- ▶ reduce tasks completed at worker are not reset

Master node failure

- ▶ Map-Reduce process is aborted

Map tasks

- ▶ usually more map tasks than nodes to improve load balancing
- ▶ it is common to use one DFS chunk per map task

Reduce tasks

- ▶ usually fewer reduce tasks than map tasks
- ▶ output of map phase is often skewed, that is, value lists for different keys differ significantly in length
- ▶ randomly assigning keys to reduce tasks usually reduces impact of skew

Pre-Aggregating Values

- ▶ Often, a map task will produce many key-value pairs with the same key.
- ▶ If the reduce function does not depend on the order of the values it receives for any particular key (that is, is commutative and associative), we can pre-aggregate values at the map worker.
- ▶ The consequence is that fewer key-value pairs enter the shuffle phase.
- ▶ This saves computation time during shuffle and, most importantly, saves network communication time.

Example (Counting words)

Instead of emitting k pairs $(w, 1)$, we let the map tasks emit one pair (w, k) .

7.2 Map-Reduce Algorithms

Matrix-Vector Multiplication

Task

Compute $A\mathbf{v}$ for matrix $A = (a_{ij})_{\substack{i \in [m] \\ j \in [n]}} \in \mathbb{R}^{m \times n}$ and vector $\mathbf{v} = (v_i)_{i \in [n]} \in \mathbb{R}^n$.

Both matrix and vector are stored in DFS.

- ▶ Matrix entries are stored as triples (i, j, a_{ij}) for $a_{ij} \neq 0$, which we may view as key-value pairs with key (i, j) .
- ▶ Vector entries are stored as pairs (i, v_i) .

Version 1: Vector \mathbf{v} fits into main memory

Each map task loads \mathbf{v} into main memory once.

MAP function: On input (i, j, a_{ij}) , emit $(i, a_{ij}v_j)$.

REDUCE function: On input $(i, values)$, emit $(i, \sum_{v \in values} v)$.

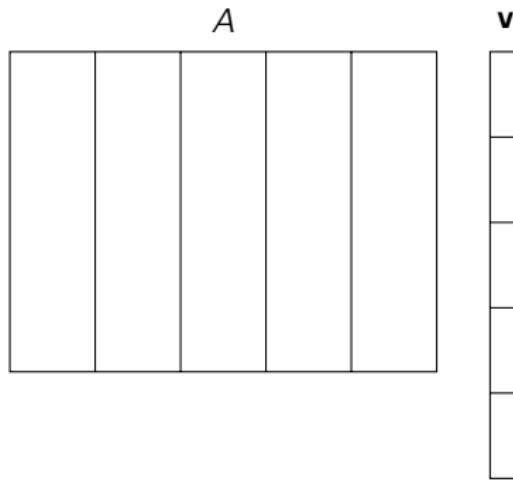
Matrix-Vector Multiplication (cont'd)

Version 2: Vector \mathbf{v} does not fit into main memory

- Problem:
- ▶ If \mathbf{v} does not fit into main memory, map tasks may have to make a large number of disk accesses.
- Solution:
- ▶ Partition \mathbf{v} into segments such that each segment fits into main memory.
 - ▶ Partition matrix into corresponding stripes.
 - ▶ Store matrix in DFS in such a way that each chunk contains only elements from one stripe.
 - ▶ Then map task loads segment of vector corresponding to matrix stripe of its chunk into main memory.
 - ▶ MAP and REDUCE functions remain the same.

Notes for Page 7.20

Partition of vector and matrix



Review of Relational Data Model

The **relational data model** uses the mathematical concept of a **relation** as the formalism for describing and representing data.

- ▶ Intuitively, a relation can be thought of as a **table**.
- ▶ Formally, it is a subset of a Cartesian product of sets.

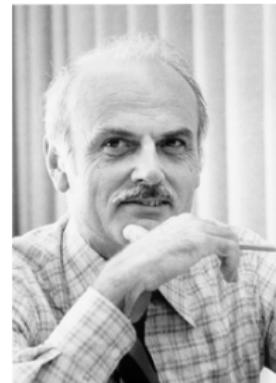


Photo: Wikipedia

Edgar C. Codd (1923–2003)

ACM Turing Award 1981

For his fundamental and continuing contributions to the theory and practice of database management systems.

Review of Relational Data Model: Relations

A *k*-ary relation is a subset of a Cartesian product of *k* sets:

$$\mathcal{R} \subseteq U_1 \times \cdots \times U_k.$$

We may view a *k*-ary relation as a **table** with *k* columns.

<i>R</i>		
a_1	\cdots	a_k
b_1	\cdots	b_k
c_1	\cdots	c_k
\vdots		\vdots

if $\mathcal{R} = \{(a_1, \dots, a_k), (b_1, \dots, b_k), (c_1, \dots, c_k), \dots\}$.

The **rows** of the table correspond to the **tuples** in the relation.

Remark 7.1

A relation is a **set** and as such unordered. The order in which the tuples (=rows) appear in the table is irrelevant and not part of the data model.

Review of Relational Data Model: Attributes and Schemas

It is useful to give names to the columns of a table. They are called **attributes**.

R		
A_1	\dots	A_k
a_1	\dots	a_k
b_1	\dots	b_k
\vdots		\vdots

A **relation schema** is an expression $R(A_1, \dots, A_k)$ consisting of a **relation name** R and a **sort** (A_1, \dots, A_k) , which is a list (A_1, \dots, A_k) of attributes.

A **database schema** is a set of relation schemas.

Review of Relational Algebra

- ▶ The **relational algebra** is based on a few basic operators that take one or two relations as input and map them to a new relation.
- ▶ These operators can be combined to **relational algebra expressions** specifying more complex mappings from database instances (that is, sets of relations) to relations: the **relational algebra queries**.
- ▶ Relational algebra is the core of the query language SQL.

Review of Relational Algebra: Operations

The main relational algebra operations are:

Selection: Apply a condition C to all tuples in a relation \mathcal{R} and only keep those satisfying C .

Thus selection deletes rows from a table.

Projection: For a subset S of the attributes of a relation \mathcal{R} , discard all entries corresponding to attributes not in S from the tuples in \mathcal{R} and return relation $\pi_S(\mathcal{R})$ consisting of the remainder of the tuples.

Thus projection deletes columns from a table.

Union, Intersection, Difference: Apply set-theoretic operation to two relations.

Review of Relational Algebra: Operations (cont'd)

Natural Join: Merge two relations $\mathcal{R}, \mathcal{R}'$ into one by taking all pairs of tuples $t \in \mathcal{R}, t' \in \mathcal{R}'$ that coincide on the common attributes of \mathcal{R} and \mathcal{R}' and form a new tuple on all attributes of \mathcal{R} and \mathcal{R}' that coincides with t and t' on their attributes. Return relation $\mathcal{R} \bowtie \mathcal{R}'$ consisting of all such tuples.

Grouping and Aggregation: Partition, or group, tuples of a relation \mathcal{R} according to their values on some of the attributes and apply aggregation operators like SUM or AVG to remaining attributes in each group.

Relational Algebra in Map-Reduce

- ▶ All relational algebra operators can easily be implemented as Map-Reduce processes.
- ▶ We only consider a few examples.

Storing relations in DFS

We assume that the tuples t of a relation \mathcal{R} of schema $R(\bar{A})$ are stored as key-value pairs (R, t) .

Projection

Assume that we have a relation \mathcal{R} of schema $R(A, B, C)$, and we want to compute the projection $Q := \pi_{A,C}(\mathcal{R})$.

MAP function: On input $(R, (a, b, c))$, emit $((a, c), 1)$.

REDUCE function: On input $((a, c), values)$, emit $(Q, (a, c))$.

Note that the projection is computed in the map phase; in the reduce phase we only remove duplicates.

Assume that we have relations \mathcal{R}, \mathcal{S} of the same sort, and we want to compute the intersection $\mathcal{Q} := \mathcal{R} \cap \mathcal{S}$.

MAP function: On input (R, t) , emit (t, R) .
On input (S, t) , emit (t, S) .

REDUCE function: On input $(t, values)$, if list $values$ contains both R and S , emit (Q, t) .

Assume we have a relation \mathcal{R} of schema $R(A, B)$ and a relation \mathcal{S} of schema $S(B, C)$, return the natural join $\mathcal{Q} = \mathcal{R} \bowtie \mathcal{S}$.

MAP function: On input $(R, (a, b))$, emit $(b, (R, a))$.
On input $(S, (b, c))$, emit $(b, (S, c))$.

REDUCE function: On input $(b, values)$, emit $(Q, (a, b, c))$ for all $(R, a), (S, c) \in values$.

Grouping and Aggregation

Assume we have a relation \mathcal{R} of schema $R(A, B, C)$, and we want to group by attribute A and take the average over the C -values. Let \mathcal{Q} denote the resulting relation.

MAP function: On input $(R, (a, b, c))$, emit (a, c) .

REDUCE function: On input $(a, values)$, compute the average c^* of the entries $c \in values$ and emit $(Q, (a, c^*))$.

Matrix Multiplications

Assume we have two matrices $A = (a_{ij}) \in \mathbb{R}^{\ell \times m}$ and $B = (b_{jk}) \in \mathbb{R}^{m \times n}$ and want to compute $C := AB \in \mathbb{R}^{\ell \times n}$.

We think of the elements of A as being stored as key-value pairs $(A, (i, j, a_{ij}))$ and the elements of B as being stored as key-value pairs $(B, (j, k, b_{jk}))$.

Idea

- ▶ Matrix multiplication is almost like a join followed by grouping and aggregation.
- ▶ View A as a ternary relation with attributes I, J, V and B as a ternary relation with attributes J, K, W .
- ▶ In a first map-reduce round, we can compute the join, giving us a 5-ary relation with tuples $(i, j, k, a_{ij}, b_{jk})$.
In fact, it is better to compute the ternary relation with tuples $(i, k, a_{ij}b_{jk})$.
- ▶ In a second map-reduce round, we group by I, K and sum over the VW -values.

Matrix Multiplications (cont'd)

First Map-Reduce Round

MAP function: On input $(A, (i, j, v))$, emit $(j, (A, i, v))$.

On input $(B, (j, k, w))$, emit $(j, (B, k, w))$.

REDUCE function: On input (j, values) , emit $((i, k), vw)$ for all $(A, i, v), (B, k, w) \in \text{values}$ such that $vw \neq 0$.

Second Map-Reduce Round

MAP function: The identity function: on input $((i, k), x)$, emit $((i, k), x)$.

REDUCE function: On input $((i, k), \text{values})$, compute the sum x^* of all $x \in \text{values}$ and emit $(C, (i, k, x^*))$.

Matrix Multiplication in one Map-Reduce Round

MAP function: On input $(A, (i, j, v))$, emit all key-value pairs $((i, k), (A, j, v))$ for $k \in [n]$.

On input $(B, (j, k, w))$, emit all key-value pairs $((i, k), (B, j, w))$ for $i \in [\ell]$.

REDUCE function: On input $((i, k), values)$, compute the sum x of all vw for $(A, j, v), (B, j, w) \in values$ and emit $(C, (i, k, x))$.

Remark 7.2

We will see later that the two-round algorithm for matrix multiplication is often the better one, especially if the matrices are sparse.

7.3 Analysis of Map-Reduce Algorithms

Wall-clock time

Total time for the MR-process to finish.

Discussion

- ▶ ultimately the parameter we are most interested in
- ▶ heavily system-dependent
- ▶ abstract model would consist of model of single nodes (standard RAM model) and model of communication channels, with parameters like bandwidth and reliability
- ▶ too complicated to analyse

Cost Measures (cont'd)

Number of rounds

Number of MR-rounds in the process

Discussion

- ▶ reasonable and important cost factor
- ▶ not sufficient alone, has to be viewed in connection with other cost measures

Cost Measures (cont'd)

Communication cost

Sum of input sizes to all tasks.

Discussion

- ▶ communication cost typically dominates the execution cost; in particular it dominates the computation time of the nodes
- ▶ we could also take the output size(s) into account, but except for the final output, they are inputs to other tasks and thus counted, and the final output is usually small (aggregated, projected, . . .), because otherwise it is useless
- ▶ input sizes can be measured in bits or more abstract measures such as number of tuples (in a relational database context)
- ▶ communication cost is a reasonable cost measure, but only for “reasonable” algorithms with good load balancing
- ▶ the communication cost can be minimised by carrying out the whole computation at one node, which of course is pointless

Cost Measures (cont'd)

Replication rate

Number of key-value pairs produced by all map tasks divided by the input size.

Discussion

- ▶ relative communication cost
- ▶ only relevant for single-round MR-processes

Maximum load (also called reducer size)

Maximum input length for single reducer or reduce task

Discussion

- ▶ measures load balancing
- ▶ has impact on execution time of reducers and hence on wall-clock time

7.4 Analysis of Matrix Multiplication

We analyse the communication cost and maximum load of our two matrix multiplication algorithms.

As before, the input matrices are $A = (a_{ij}) \in \mathbb{R}^{\ell \times m}$ and $B = (b_{jk}) \in \mathbb{R}^{m \times n}$.

Assumption

The non-zero entries of the two matrices are randomly distributed.

More precisely, we assume that

- ▶ $\Pr(a_{ij} \neq 0) = p$, independently for all i, j ,
- ▶ $\Pr(b_{jk} \neq 0) = q$, independently for all j, k ,

for (presumably small) p, q with $0 \leq p, q \leq 1$.

Communication Cost

Two-round: Expected communication cost

$$2p\ell m + 2qmn + 2pq\ell mn.$$

Single-round: Expected communication cost

$$p\ell m + qmn + (p + q)\ell mn.$$

Maximum Load

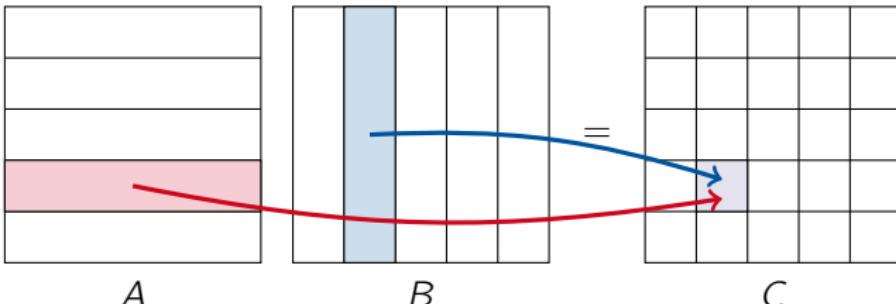
- Two-round: Expected load of reducer in the first round is $p\ell + qn$; with high probability the maximum load is below $(1 + \varepsilon)(p\ell + qn)$.
Expected load of reducer in the second round is pqm ; with high probability the maximum load is below $(1 + \varepsilon)pqm$.
- Single-round: Expected load of reducer is $pm + qm$; with high probability the maximum load is below $(1 + \varepsilon)(pm + qm)$.

Discussion

- ▶ For sparse matrices, the communication cost of the two-round algorithm is likely to be much lower.
- ▶ The maximum load of both algorithms is comparable.
- ▶ We will now see another Map-Reduce algorithm that uses fewer reducers, improving the communication cost.

Generalised Single Round Matrix Multiplication

For simplicity, we assume that we multiply square matrices $A, B \in \mathbb{R}^{n \times n}$.



Idea

- ▶ Split the matrices in stripes of equal width and use pairs of indices of the stripes (and not the rows and columns) as keys for the reducer.
- ▶ This reduces the replication rate (and thus the communication cost), because it means that a single entry of the matrix only has to be sent to each stripe of the other matrix and not to each row/column.

Generalised Single Round Algorithm

Choose parameter $s = \text{number of stripes}$.

Let $h : [n] \rightarrow [s]$ be the mapping that assigns each row/column index to its stripe. For example, $h(i) = \lceil is/n \rceil$.

MAP function: On input $(A, (i, j, v))$, emit all key-value pairs $((h(i), u), (A, i, j, v))$ for $u \in [s]$.

On input $(B, (j, k, w))$, emit all key-value pairs $((t, h(k)), (B, j, k, w))$ for $t \in [s]$.

REDUCE function: On input $((t, u), values)$, for all $i \in h^{-1}(t)$ and all $k \in h^{-1}(u)$, compute the sum

$$c_{ik} := \sum_{(A, i, j, v), (B, j, k, w) \in values} vw$$

and emit $(C, (i, k, c_{ik}))$.

Special Cases

- ▶ For $s = n$, this is just our previous single-round matrix multiplication algorithm.
- ▶ For $s = 1$, it is essentially a sequential matrix multiplication algorithm where all the work is done at one reducer.

Analysis (Worst Case)

For simplicity, we assume that s is a divisor of n .

Replication rate and communication cost

The replication rate of the algorithm is s .

It follows that the communication cost is

$$2n^2 + 2sn^2 = O(sn^2)$$

in the worst case.

Maximum load

Each reducer gets all entries of n/s rows and n/s columns. Thus the maximum load is

$$\frac{2n^2}{s}.$$

Analysis (Average Case)

As before, we assume that non-zero entries of the two matrices are randomly distributed:

- ▶ $\Pr(a_{ij} \neq 0) = p$, independently for all i, j ,
- ▶ $\Pr(b_{jk} \neq 0) = q$, independently for all j, k ,

Replication rate and communication cost

The replication rate of the algorithm is s (as in the worst case).

It follows that the expected communication cost is

$$(p + q)n^2 + s(p + q)n^2 = O(s(p + q)n^2).$$

Expected load

Each reducer gets all nonzero entries of n/s rows and n/s columns.
Thus the expected load is

$$\frac{(p + q)n^2}{s}.$$

7.5 Multiway Joins in Map Reduce

Task

Compute the natural join $\mathcal{Q} = \mathcal{R}_1 \bowtie \cdots \bowtie \mathcal{R}_\ell$.

Let

- ▶ $R_i(A_{i1}, \dots, A_{ik_i})$ be the schema of \mathcal{R}_i ,
- ▶ A_1, \dots, A_k be the list of all attributes of all the relations, that is,

$$\{A_1, \dots, A_k\} = \{A_{ij} \mid i \in [\ell], j \in [k_i]\}$$

- ▶ V_i be the set of values of attribute A_i

Parameters of the algorithm

- ▶ s = number of reducers
- ▶ $s_1, \dots, s_k \in \mathbb{N}$ such that $\prod_{i=1}^k s_i = s$
 s_i is called the **share** of attribute A_i
- ▶ h_1, \dots, h_k independently chosen hash functions $h_i : V_i \rightarrow [s_i]$ (for simplicity assumed to be truly random)

The Hypercube Algorithm

MAP function: On input $(R_i, (a_1, \dots, a_{k_i}))$, emit all pairs

$$\left((p_1, \dots, p_k), (R_i, (a_1, \dots, a_{k_i})) \right)$$

such that

- ▶ $p_j \in [s_j]$ for all $j \in [k]$,
- ▶ $p_j = h_j(a_{j'})$ for all $j \in [k], j' \in [k_i]$ such that $A_{ij'} = A_j$.

REDUCE function: On input $(\bar{p}, values)$, compute

$$\mathcal{Q}(\bar{p}) := \mathcal{R}_1(\bar{p}) \bowtie \dots \bowtie \mathcal{R}_\ell(\bar{p}),$$

where

$$\mathcal{R}_i(\bar{p}) := \{t \mid (R_i, t) \in values\},$$

and emit all pairs (Q, t) for $t \in \mathcal{Q}(\bar{p})$.

Example.

We consider the natural join

$$Q = R_1 \bowtie R_2 \bowtie R_3,$$

with

- ▶ attributes A_1, A_2, A_3 with domains $V_1 = [120], V_2 = V_3 = [60]$.
- ▶ relation schemas $R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_3, A_1)$ for R_1, R_2, R_3 , respectively.

We assume we have

$$s = 24$$

reducers and let

$$s_1 := 4, \quad s_2 := 3, \quad s_3 := 2.$$

We define the hash functions $h_i : V_i \rightarrow [s_i]$ by

$$h_1(x) := \left\lceil \frac{x}{30} \right\rceil, \quad h_2(x) := \left\lceil \frac{x}{20} \right\rceil, \quad h_3(x) := \left\lceil \frac{x}{30} \right\rceil.$$

Now suppose we have the following relations:

$$\mathcal{R}_1 = \begin{array}{|c|c|} \hline & R_1 \\ \hline A_1 & A_2 \\ \hline \end{array} \begin{array}{|c|c|} \hline & R_2 \\ \hline A_2 & A_3 \\ \hline \end{array} \begin{array}{|c|c|} \hline & R_3 \\ \hline A_3 & A_1 \\ \hline \end{array}$$

67 48
56 48
119 1
119 26

48 13
48 59
26 13

13 67
13 56
13 119
59 119

The MAP function receives as input key-value pairs $(R, (a, b))$ and emits a list of key value pairs $((p_1, p_2, p_3), (R, a, b))$.

For example,

- ▶ on input $(R_1, (67, 48))$, MAP emits
$$((3, 3, 1), (R_1, (67, 48))), \quad ((3, 3, 2), (R_1, (67, 48))).$$

To understand this, note that $h_1(67) = 3$ and $h_2(48) = 3$;

- ▶ on input $(R_2, (26, 13))$, MAP emits
$$((1, 2, 1), (R_2, (26, 13))), \quad ((2, 2, 1), (R_2, (26, 13))), \\ ((3, 2, 1), (R_2, (26, 13))), \quad ((4, 2, 1), (R_2, (26, 13))).$$

The function REDUCE receives for each key $(p_1, p_2, p_3) \in [4] \times [3] \times [2]$ a list of values of the form (R, a, b) .

For example, for the key $(3, 3, 1)$ it receives the following values:

$$(R_1, 67, 48), \quad (R_2, 48, 13), \quad (R_3, 13, 67).$$

Joining these tuples, REDUCE emits the output tuple

$$(Q, (67, 48, 13)).$$

Lemma 7.3

The Hypercube algorithm correctly computes $\mathcal{R}_1 \bowtie \cdots \bowtie \mathcal{R}_\ell$.

Proof.

Let

$$\mathcal{Q} := \mathcal{R}_1 \bowtie \dots \bowtie \mathcal{R}_\ell.$$

We need to prove that

$$\mathcal{Q} = \bigcup_{\bar{p} \in [s_1] \times \dots \times [s_k]} \mathcal{Q}(\bar{p}). \quad (*)$$

The inclusion “ \supseteq ” is clear, because for each \bar{p} and each $i \in [\ell]$ we have $\mathcal{R}_i(\bar{p}) \subseteq \mathcal{R}_i$. This implies $\mathcal{Q}(\bar{p}) \subseteq \mathcal{Q}$.

For the converse inclusion “ \subseteq ”, let $t := (a_1, \dots, a_k) \in \mathcal{Q}$. For $j = 1, \dots, k$, let $p_j := h_j(a_j)$ and $\bar{p} = (p_1, \dots, p_k)$. We shall prove that

$$t \in \mathcal{Q}(\bar{p}).$$

For $i \in [\ell]$, let $t_i := (a_{i1}, \dots, a_{ik_i})$ be the projection of the tuple t on $(A_{i1}, \dots, A_{ik_i})$. That is, $a_{ij'} = a_j$ for the $j \in [k]$ such that $A_{ij'} = A_j$. Then on input (R_i, t_i) , the MAP function emits

$$\left(\bar{p}, (R_i, (a_{i1}, \dots, a_{ik_i})) \right).$$

This implies $t_i \in \mathcal{R}_i(\bar{p})$, and as this holds for all $i \in [\ell]$, we have $t \in \mathcal{Q}(\bar{p})$. \square

Replication Rate

For every $i \in [\ell]$, we let

- ▶ $\text{Idx}(i) := \{j \in [k] \mid A_j \in \{A_{i1}, \dots, A_{ik_i}\}\}$ (the set of indices of the attributes of \mathcal{R}_i)
- ▶ $m_i := |\mathcal{R}_i|$, for $i \in [\ell]$.

We assume that the relation sizes m_i are large compared to $s = \prod_{j=1}^k s_j$.

Observation 7.4

The replication rate of the Hypercube algorithm is

$$\frac{\sum_{i=1}^{\ell} \left(m_i \cdot \prod_{j \in [k] \setminus \text{Idx}(i)} s_j \right)}{\sum_{i=1}^{\ell} m_i}.$$

Proof.

Each tuple $t \in \mathcal{R}_i$ is sent to all keys $\bar{p} = (p_1, \dots, p_k)$ where

- ▶ $p_j = h_j(a_j)$ for $j \in \text{Idx}(i)$, where a_j is the A_j -value of t ;
- ▶ $p_j \in [s_j]$ for all $j \in [k] \setminus \text{Idx}(i)$.

Hence t is sent to $\prod_{j \in [k] \setminus \text{Idx}(i)} s_j$ keys. As there are m_i tuples in \mathcal{R}_i , the number of key-value pairs emitted by all map tasks is

$$\sum_{i=1}^{\ell} \left(m_i \cdot \prod_{j \in [k] \setminus \text{Idx}(i)} s_j \right).$$

This needs to be divided by the input size $\sum_{i=1}^{\ell} m_i$. □

Expected and Maximum Load

Theorem 7.5

- (1) *The expected load of the algorithm (over the random choices of the hash functions) is*

$$\sum_{i \in [\ell]} \frac{m_i}{\prod_{j \in \text{Idx}(i)} s_j}.$$

- (2) *With high probability, the maximum load is*

$$O \left(\sum_{i \in [\ell]} \frac{m_i}{\min_{j \in \text{Idx}(i)} s_j} \right)$$

Notes for Page 7.55

Proof of (1).

Consider the reducer for the key $\bar{p} = (p_1, \dots, p_k)$. The load of this reducer is

$$\sum_{i \in [\ell]} |\mathcal{R}_i(\bar{p})|.$$

Let $i \in [\ell]$ and $t \in R_i$. For $j \in \text{Idx}(i)$, let a_j be the A_j -value of t . Then

$$\begin{aligned}\Pr_{h_1, \dots, h_k} (t \in \mathcal{R}_i(\bar{p})) &= \Pr_{h_1, \dots, h_k} (h_j(a_j) = p_j \text{ for all } j \in \text{Idx}(i)) \\ &= \prod_{j \in \text{Idx}(i)} \Pr_{h_j} (h_j(a_j) = p_j) \\ &= \prod_{j \in \text{Idx}(i)} \frac{1}{s_j}.\end{aligned}$$

Thus

$$\mathbb{E}(|\mathcal{R}_i(\bar{p})|) = \sum_{t \in \mathcal{R}_i} \Pr_{h_1, \dots, h_k} (t \in \mathcal{R}_i(\bar{p})) = \frac{m_i}{\prod_{j \in \text{Idx}(i)} s_j}.$$

This proves (1). □

We omit the proof of (2).

Skew-Free Relations

Let $i \in [\ell]$ and $J \subseteq \text{Idx}(i)$. The **frequency** of an $(A_j \mid j \in J)$ -tuple t in \mathcal{R}_i is the number of tuples $t' \in \mathcal{R}_i$ whose projection on $(A_j \mid j \in J)$ is t .

Relation \mathcal{R}_i is **skew-free** with respect to s_1, \dots, s_k , if for every set $J \subseteq \text{Idx}(i)$ and every $(A_j \mid j \in J)$ -tuple t , the frequency of t in \mathcal{R}_i is at most

$$\frac{m_i}{\prod_{j \in J} s_j}.$$

Theorem 7.6

If the relations $\mathcal{R}_1, \dots, \mathcal{R}_\ell$ are skew-free with respect to s_1, \dots, s_k , then with high probability, the maximum load is

$$O\left(\sum_{i \in [\ell]} \frac{m_i}{\prod_{j \in \text{Idx}(i)} s_j} \log^k(s)\right).$$

(Proof omitted.)

Choosing the Shares

- ▶ We want to choose the shares s_j in such a way that we minimise the maximum load.
- ▶ By the theorems, we essentially need to minimise

$$\sum_{i \in [\ell]} \frac{m_i}{\prod_{j \in \text{Idx}(i)} s_j}$$

subject to the constraint $\prod_{j=1}^k s_j = s$.

- ▶ This is a difficult optimisation problem, especially since we look for an integer solution. However, there are several techniques for obtaining an optimal real solution, and then rounding usually gives us a reasonable integer solution.
- ▶ We assume that we know the sizes m_i of the input relations. Sometimes, we have more statistical information about the relation, and we can exploit this as well to compute good shares s_i to minimise the maximum load.

Example

Suppose we want to join relations

- ▶ \mathcal{R}_1 of schema $R(A_1, A_2)$,
- ▶ \mathcal{R}_2 of schema $S(A_2, A_3)$,
- ▶ \mathcal{R}_3 of schema $T(A_1, A_3)$,

and we have

$$m_1 = |\mathcal{R}_1| = 10^{11}, \quad m_2 = |\mathcal{R}_2| = 10^{10}, \quad m_3 = |\mathcal{R}_3| = 10^{11}.$$

Suppose furthermore that we want to use $s = 10^4$ reducers.

We define the shares of the attributes A_1, A_2, A_3 as follows:

$$s_1 := 10^2, \quad s_2 := s_3 := 10.$$

Then the replication rate of the hypercube algorithm with these parameters is

$$\frac{10^{11} \cdot 10 + 10^{10} \cdot 100 + 10^{11} \cdot 10}{10^{11} + 10^{10} + 10^{11}} = \frac{3 \cdot 10^{12}}{2 \cdot 10^{11} + 10^{10}} = \frac{100}{7} \approx 14.3.$$

The expected load of the reducers is

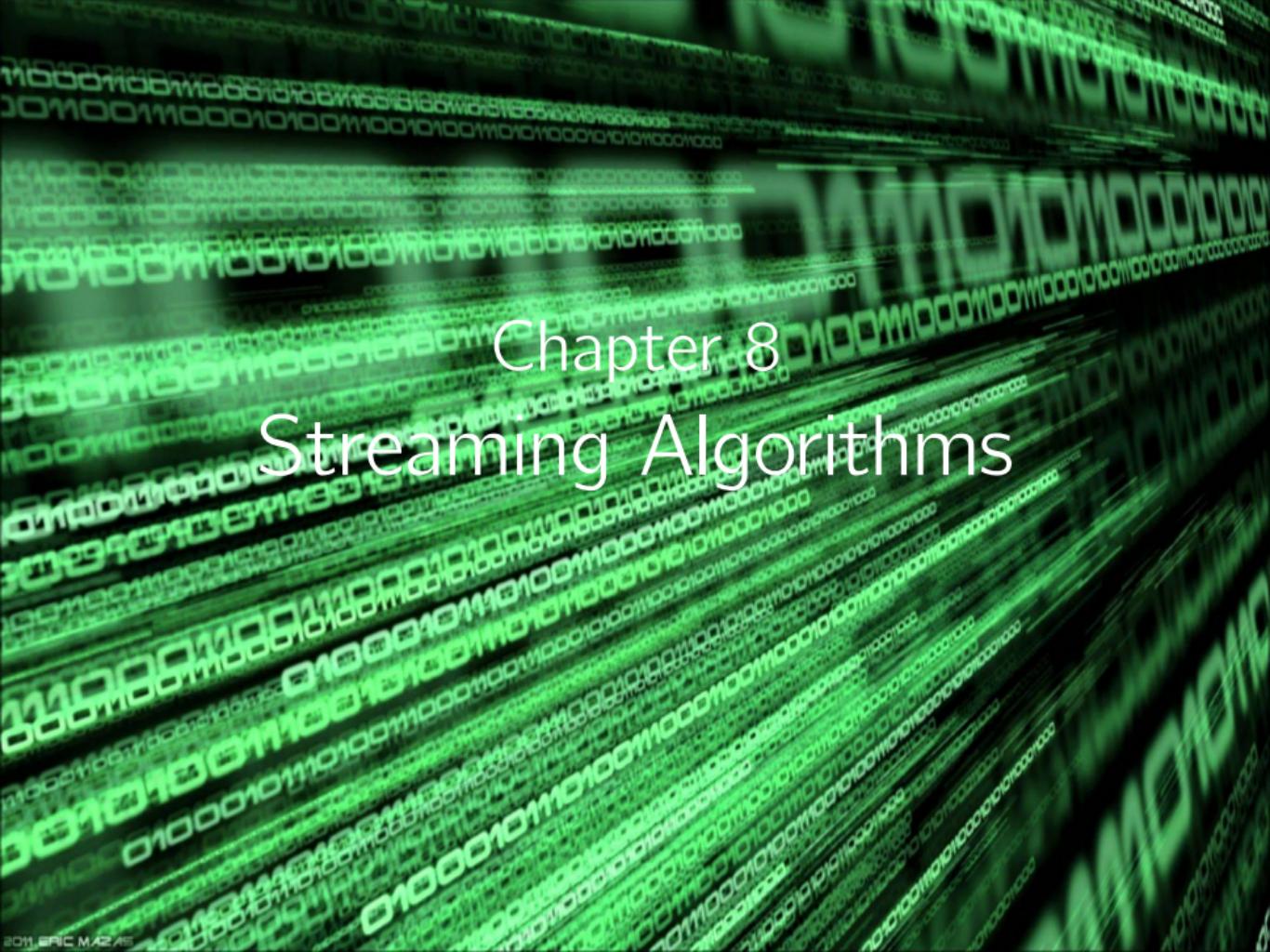
$$\frac{10^{11}}{10^3} + \frac{10^{10}}{10^2} + \frac{10^{11}}{10^3} = 3 \cdot 10^8.$$

References

This chapter is based on [Leskovec et al., 2014, Chapter 2].

Chapter 8

Streaming Algorithms



8.1 Basics

Scenario

- ▶ data items arrive in a **stream** a_1, a_2, \dots
- ▶ data volume too large to store everything in main (random access) memory

Examples 8.1

- ▶ sensor data
- ▶ IP addresses observed by router
- ▶ incoming emails at mail server

Goal

Design of efficient (sublinear space, online, real-time) algorithms for data analysis tasks

- ▶ Data items from some universe \mathbb{U} .
- ▶ We let $N := |\mathbb{U}|$ be the size of the universe. Sometimes, it is convenient to assume $\mathbb{U} = \{0, \dots, N - 1\}$.
- ▶ Input data stream is

$$a_1, \dots, a_n$$

where $a_i \in \mathbb{U}$.

- ▶ The length n of the data stream is not known to the algorithm in advance.
- ▶ n may appear as a parameter in the analysis of the algorithm.
Typically, we want algorithms using space

$$\text{polylog}(n + N) = \bigcup_{k \geq 1} \log(n + N)^k$$

(assuming $n + N \geq 2$).

Sampling Uniformly from the Stream

Task

Pick element a_i for i uniformly at random from $[n]$.

Remark 8.2

Trivial if n is known in advance.

Simple Sampling Algorithm

Algorithm SIMPLESAMPLE

Input: Stream a_1, \dots, a_n

▷ Assume $n \geq 1$

1. $i \leftarrow 0$
2. while not end of stream do
3. $i \leftarrow i + 1$
4. $sample \leftarrow a_i$ with probability $1/i$
 ▷ otherwise $sample$ keeps its current value
5. return $sample$

Theorem 8.3

The index of the element returned by the algorithm on input a_1, \dots, a_n is i with probability $1/n$.

Thus the algorithm returns $a \in \mathbb{U}$ with probability

$$\frac{|\{i \mid a_i = a\}|}{n}.$$

Proof.

Let x_i be the index of the last element assigned to *sample* after the i th iteration of the loop.

By induction on $i \geq 1$ we prove that for all $j \in [i]$ we have

$$\Pr(x_i = j) = \frac{1}{i}.$$

$i = 1$: After the first iteration, we have $x_1 = 1$ with probability 1.

$i \rightarrow i + 1$: After the $(i + 1)$ st iteration, we have:

- ▶ $\Pr(x_{i+1} = i + 1) = \frac{1}{i + 1}$
- ▶ and for $1 \leq j \leq i$,

$$\Pr(x_{i+1} = j) = \Pr(x_i = j) \cdot \left(1 - \frac{1}{i+1}\right) = \frac{1}{i} \cdot \frac{i}{i+1} = \frac{1}{i+1}. \quad \square$$

Reservoir Sampling

Task

Pick elements a_{i_1}, \dots, a_{i_k} for $\{i_1, \dots, i_k\}$ uniformly at random from $\binom{[n]}{k}$.

That is, compute a “reservoir” of k randomly chosen elements of the stream.

Sampling Algorithm

Algorithm RESERVOIRSAMPLE

Input: Stream a_1, \dots, a_n , $k \leq n$

1. for $i = 1, \dots, k$ do
2. $sample[i] \leftarrow a_i$ ▷ variable i has value k now
3. while not end of stream do
4. $i \leftarrow i + 1$
5. $replace \leftarrow \begin{cases} \text{true} & \text{with probability } \frac{k}{i}, \\ \text{false} & \text{otherwise} \end{cases}$
6. if $replace$ then
7. choose j uniformly at random from $[k]$
8. $sample[j] \leftarrow a_i$
9. return $sample$

Theorem 8.4

The set of indices of the elements returned by the algorithm on input a_1, \dots, a_n and $k \leq n$ is $\{i_1, \dots, i_k\} \in \binom{[n]}{k}$ with probability $\frac{1}{\binom{n}{k}}$.

Space requirement

The algorithm needs space

$$O(\log n + k \cdot \log N).$$

Proof of the theorem.

For $i > k$, let X_i be the set of indices of the sampled elements after the $(i - k)$ th iteration of the loop, and $X_k := \{1, \dots, k\}$ the set of indices before the first iteration.

By induction on $i \geq k$ we prove that for all $J \in \binom{[i]}{k}$ we have

$$\Pr(X_i = J) = \binom{i}{k}^{-1}$$

$i = k$: We have $X_k = \{1, \dots, k\}$ with probability $1 = \binom{k}{k}^{-1}$.

$i \rightarrow i + 1$: Let $J \in \binom{[i+1]}{k}$.

Case 1: $i + 1 \in J$.

For each set $J' \in \binom{[i]}{k}$, we have

$$\Pr(X_{i+1} = J \mid X_i = J') = \begin{cases} \frac{k}{i+1} \cdot \frac{1}{k} = \frac{1}{i+1} & \text{if } |J \cap J'| = k - 1, \\ 0 & \text{otherwise.} \end{cases}$$

Thus

$$\begin{aligned}\Pr(X_{i+1} = J) &= \sum_{J' \in \binom{[i]}{k}} \Pr(X_{i+1} = J \wedge X_i = J') \\&= \sum_{J' \in \binom{[i]}{k}} \Pr(X_{i+1} = J \mid X_i = J') \cdot \Pr(X_i = J') \\&= \binom{i}{k}^{-1} \sum_{J' \in \binom{[i]}{k}} \Pr(X_{i+1} = J \mid X_i = J') \quad (\text{I.H.}) \\&= \binom{i}{k}^{-1} \sum_{J' \in \binom{[i]}{k} \text{ with } |J \cap J'|=k-1} \frac{1}{i+1} \\&= \binom{i}{k}^{-1} \cdot \frac{(i - (k-1))}{i+1} \\&= \binom{i+1}{k}^{-1}\end{aligned}$$

Case 2: $i + 1 \notin J$.

Then

$$\begin{aligned}\Pr(X_{i+1} = J) &= \Pr(X_i = J) \cdot \left(1 - \frac{k}{i+1}\right) \\ &= \binom{i}{k}^{-1} \cdot \frac{(i-k+1)}{i+1} \quad (\text{I.H.}) \\ &= \binom{i+1}{k}^{-1}.\end{aligned}$$

□

8.2 Hashing

Hash Functions

- ▶ A **hash function** on a universe \mathbb{U} is simply a function from \mathbb{U} to a set \mathbb{T} , which is usually an initial segment of the natural numbers.
Implicitly, we always assume that a hash function is or at least looks random.
- ▶ Formally, we consider a probability distribution on the space of all functions $h : \mathbb{U} \rightarrow \mathbb{T}$.
- ▶ If this probability distribution is the uniform distribution, then we are dealing with **truly random** hash functions.
- ▶ Often, the analysis of algorithms based on hashing relies on the assumption that we have truly random hash functions.
- ▶ However, unless the size N of the universe is small, in which case we normally need no hashing in the first place, this assumption is unrealistic.

The range \mathbb{T} of a hash function

We usually assume that $\mathbb{T} = \{0, \dots, M - 1\}$ for some positive $M \in \mathbb{N}$.

Sometimes it is convenient to assume that \mathbb{T} is some other set, such as $[M]$ or $\{1, -1\}$. Then we can fix a bijection $f : \{0, \dots, |\mathbb{T}| - 1\} \rightarrow \mathbb{T}$ and choose hash functions $h = f \circ h'$, where h' is a hash function with range $\{0, \dots, |\mathbb{T}| - 1\}$.

Why is True Randomness Useful?

We consider hash functions $h : \mathbb{U} \rightarrow \mathbb{T}$, where $|\mathbb{T}| = M$.

Suppose we choose h uniformly at random from the class $\mathbb{T}^{\mathbb{U}}$ of all functions from \mathbb{U} to \mathbb{T} .

Then we have the following useful properties.

- ▶ For all $x \in \mathbb{U}, y \in \mathbb{T}$ we have

$$\Pr_{h \in \mathbb{T}^{\mathbb{U}}} (h(x) = y) = \frac{1}{M}.$$

- ▶ For all distinct $x, x' \in \mathbb{U}$ we have

$$\Pr_{h \in \mathbb{T}^{\mathbb{U}}} (h(x) = h(x')) = \frac{1}{M}.$$

- ▶ For all distinct $x_1, \dots, x_k \in \mathbb{U}$ and all $y_1, \dots, y_k \in \mathbb{T}$ we have

$$\Pr_{h \in \mathbb{T}^{\mathbb{U}}} (h(x_1) = y_1 \wedge \dots \wedge h(x_k) = y_k) = \frac{1}{M^k}.$$

(These are just examples, there are many more)

Why is True Randomness not Practical?

Number of random bits.

Randomness is a scarce resource, and to generate a random function from $\{0, \dots, N - 1\}$ to $\{0, \dots, M - 1\}$ we need $\Theta(N \log M)$ random bits.

Space requirement.

To store a random function from $\{0, \dots, N - 1\}$ to $\{0, \dots, M - 1\}$ we need an expected number of $\Theta(N \log M)$ bits.

In typical applications, N is very large, much larger than the main memory. Thus the space requirement alone is prohibitive.

Families of Hash Functions

Feasible distributions of hash functions are usually obtained by fixing a small (compared to the set of all functions from \mathbb{U} to \mathbb{T}) family \mathcal{H} of hash functions from \mathbb{U} to \mathbb{T} and considering the uniform distribution on this family.

We write

$$\Pr_{h \in \mathcal{H}} (\dots)$$

to denote that h is drawn uniformly at random from \mathcal{H} .

Definition 8.5

A family \mathcal{H} of hash functions from \mathbb{U} to \mathbb{T} is **universal** if for all distinct $x, x' \in \mathbb{U}$,

$$\Pr_{h \in \mathcal{H}} (h(x) = h(x')) \leq \frac{1}{|\mathbb{T}|}.$$

Example 8.6

Let $M \leq N$, and let $p \geq N$ be a prime. Suppose that $\mathbb{U} = \{0, \dots, N-1\}$ and $\mathbb{T} = \{0, \dots, M-1\}$. For $a, b \in \mathbb{N}$, define $h_{a,b} : \mathbb{U} \rightarrow \mathbb{T}$ by

$$h_{a,b}(x) := ((ax + b) \bmod p) \bmod M$$

Then the family $\mathcal{H} := \{h_{a,b} \mid a, b \in \{0, \dots, p-1\}, a \neq 0\}$ is universal.

Proof that \mathcal{H} is universal.

Let $x_1, x_2 \in \mathbb{U}$ be distinct.

Claim

For all $c_1, c_2 \in \{0, \dots, p - 1\}$ there is exactly one pair $a, b \in \{0, \dots, p - 1\}$ such that $ax_1 + b = c_1 \bmod p$ and $ax_2 + b = c_2 \bmod p$. Furthermore, $a \neq 0$ if and only if $c_1 \neq c_2$.

Proof.

We regard the equations $ax_1 + b = c_1 \bmod p$ and $ax_2 + b = c_2 \bmod p$ as a system of linear equations in the two variables a, b over the p -element field \mathbb{F}_p . We can write this system as

$$\underbrace{\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \end{pmatrix}}_{=:A} \begin{pmatrix} a \\ b \end{pmatrix} = \underbrace{\begin{pmatrix} c_1 \\ c_2 \end{pmatrix}}_{=:c}.$$

The matrix A of this system is nonsingular, and thus the system has the unique solution $A^{-1}\mathbf{c}$. Calculation shows

$$a = \frac{c_2 - c_1}{x_2 - x_1} \bmod p, \quad b = c_1 - ax_1.$$

Note that $a \neq 0 \iff c_1 \neq c_2$. \(\square\)

It follows from the claim that for all $c_1, c_2 \in \{0, \dots, p-1\}$ with $c_1 \neq c_2$,

$$\Pr_{\substack{a,b \in \{0, \dots, p-1\} \\ a \neq 0}} (ax_1 + b = c_1 \bmod p \wedge ax_2 + b = c_2 \bmod p) = \frac{1}{p(p-1)}$$

Observe: For $a, b \in \{0, \dots, p-1\}$ with $a \neq 0$ we have

$$h_{a,b}(x_1) = h_{a,b}(x_2)$$

if and only if there are $c_1, c_2 \in \{0, \dots, p-1\}$ with $c_1 \neq c_2$ and $c_1 = c_2 \bmod M$ such that

$$ax_1 + b = c_1 \bmod p \quad \text{and} \quad ax_2 + b = c_2 \bmod p.$$

For each $c_1 \in \{0, \dots, p-1\}$ there are at most $\lceil p/M \rceil - 1$ elements $c_2 \in \{0, \dots, p-1\}$ such that $c_1 \neq c_2$ and $c_1 = c_2 \pmod M$. Thus the number of pairs $c_1, c_2 \in \{0, \dots, p-1\}$ with $c_1 \neq c_2$ and $c_1 = c_2 \pmod M$ is at most

$$p(\lceil p/M \rceil - 1).$$

Observe that $\lceil p/M \rceil \leq \frac{p+M-1}{M}$ and thus

$$p\left(\left\lceil \frac{p}{M} \right\rceil - 1\right) \leq \frac{p(p+M-1)}{M} - p = \frac{p(p-1)}{M}.$$

It follows that

$$\Pr_{h \in \mathcal{H}}(h(x_1) = h(x_2)) \leq \frac{p(\lceil p/M \rceil - 1)}{p(p-1)} \leq \frac{\frac{p(p-1)}{M}}{p(p-1)} = \frac{1}{M}.$$



Problem

We want to assign *k-bit signatures* to the elements of an n -element subset $S \subseteq \mathbb{U}$ in such a way that we have few collisions, that is, pairs of distinct elements from S that get the same signature.

For a function $h : \mathbb{U} \rightarrow \mathbb{T}$ and a set $S \subseteq \mathbb{U}$, we let

$$\text{coll}(h, S) := \left| \left\{ \{x, x'\} \in \binom{S}{2} \mid h(x) = h(x') \right\} \right|$$

set of 2-element
subsets of S

denote the number of collisions of h on S .

Remark 8.7

These “signatures” merely serve as small representations of the elements of set S . They have nothing to do with signatures in a cryptographic sense.

Signatures (cont'd)

Lemma 8.8

Let \mathcal{H} be a universal family of hash functions from \mathbb{U} to $\{0, \dots, 2^k - 1\}$. Then for every $\delta > 0$ and every set $S \subseteq \mathbb{U}$ of cardinality $|S| = n$,

$$\mathsf{E}_{h \in \mathcal{H}} (\text{coll}(h, S)) = \frac{n(n-1)}{2^{k+1}}$$

and $\Pr_{h \in \mathcal{H}} \left(\text{coll}(h, S) \geq \frac{n^2}{\delta 2^{k+1}} \right) \leq \delta$.

Theorem 8.9

Let $n \in \mathbb{N}$ and $\delta > 0$. Let \mathcal{H} be a universal family of hash functions from \mathbb{U} to $\{0, \dots, 2^k - 1\}$, where $k \geq 2 \log n + \log \frac{1}{\delta} - 1$. Then for every set $S \subseteq \mathbb{U}$ of cardinality $|S| \leq n$,

$$\Pr_{h \in \mathcal{H}} (\text{coll}(h, S) \geq 1) \leq \delta.$$

Proof of the lemma.

For all distinct $w, x \in \mathbb{U}$, let X_{wx} be the indicator random variable that $h(w) = h(x)$. Then

$$\Pr_{h \in \mathcal{H}}(X_{wx} = 1) = \Pr_{h \in \mathcal{H}}(h(w) = h(x)) \leq \frac{1}{2^k},$$

because \mathcal{H} is universal.

Let X be the random variable defined by $X(h) := \text{coll}(h, S)$, for h drawn uniformly at random from \mathcal{H} . Then

$$X = \sum_{\{w,x\} \in \binom{S}{2}} X_{wx}$$

and thus

$$\mathbb{E}(X) \leq \binom{n}{2} \cdot \frac{1}{2^k} = \frac{n(n-1)}{2^{k+1}} < \frac{n^2}{2^{k+1}}$$

By Markov's inequality,

$$\Pr_{h \in \mathcal{H}} \left(\text{coll}(h, S) \geq \frac{n^2}{\delta 2^{k+1}} \right) \leq \Pr_{h \in \mathcal{H}} \left(X \geq \frac{\mathbb{E}(X)}{\delta} \right) \leq \delta.$$



Proof of the theorem.

By the lemma, we need to prove

$$1 \geq \frac{n^2}{\delta 2^{k+1}}.$$

We have

$$\begin{aligned} k &\geq 2 \log n + \log \frac{1}{\delta} - 1 \\ \implies k + 1 &\geq \log \left(\frac{n^2}{\delta} \right) \\ \implies 2^{k+1} &\geq \frac{n^2}{\delta} \\ \implies 1 &\geq \frac{n^2}{\delta 2^{k+1}}. \end{aligned}$$



(Strongly) k -Universal Families

Definition 8.10

Let $k \geq 2$, and let \mathcal{H} be a family of hash functions from \mathbb{U} to \mathbb{T} .

- (1) \mathcal{H} is **k -universal** if for all distinct $x_1, \dots, x_k \in \mathbb{U}$,

$$\Pr_{h \in \mathcal{H}} (h(x_1) = h(x_2) = \dots = h(x_k)) \leq \frac{1}{|\mathbb{T}|^{k-1}}$$

- (2) \mathcal{H} is **strongly k -universal** if for all distinct $x_1, \dots, x_k \in \mathbb{U}$ and all $y_1, \dots, y_k \in \mathbb{T}$,

$$\Pr_{h \in \mathcal{H}} (h(x_1) = y_1 \wedge \dots \wedge h(x_k) = y_k) = \frac{1}{|\mathbb{T}|^k}$$

Observation 8.11

- (1) \mathcal{H} is 2-universal if and only if it is universal.
(2) If \mathcal{H} is strongly k -universal then it is k -universal.

Proof of the Observation.

(1) is obvious.

For (2), note that

$$\begin{aligned}\Pr_{h \in \mathcal{H}} (h(x_1) = h(x_2) = \dots = h(x_k)) &= \sum_{y \in \mathbb{T}} \Pr_{h \in \mathcal{H}} (h(x_1) = h(x_2) = \dots = h(x_k) = y) \\ &= |\mathbb{T}| \cdot \frac{1}{|\mathbb{T}|^k}\end{aligned}$$

if \mathcal{H} is strongly k -universal.

□

An Alternative Characterisation of Strongly k -Universal Families

Lemma 8.12

Let $2 \leq k \leq |\mathbb{U}|$, and let \mathcal{H} be a family of hash functions from \mathbb{U} to \mathbb{T} . Then \mathcal{H} is strongly k -universal if and only if it has the following two properties.

k-Independence: For all distinct $x_1, \dots, x_k \in \mathbb{U}$ and all $y_1, \dots, y_k \in \mathbb{T}$,

$$\Pr_{h \in \mathcal{H}} \left(\bigwedge_{i=1}^k h(x_i) = y_i \right) = \prod_{i=1}^k \Pr_{h \in \mathcal{H}} (h(x_i) = y_i).$$

That is, the indicator random variables for the events $h(x_i) = y_i$ are independent.

Uniformity: For all $x \in \mathbb{U}$ and $y \in \mathbb{T}$,

$$\Pr_{h \in \mathcal{H}} (h(x) = y) = \frac{1}{|\mathbb{T}|}$$

Proof.

The backward direction is straightforward: for distinct $x_1, \dots, x_k \in \mathbb{U}$ and $y_1, \dots, y_k \in \mathbb{T}$ we have

$$\begin{aligned} \Pr_{h \in \mathcal{H}} \left(\bigwedge_{i=1}^k h(x_i) = y_i \right) &= \prod_{i=1}^k \Pr_{h \in \mathcal{H}} (h(x_i) = y_i) && \text{by } k\text{-Independence} \\ &= \prod_{i=1}^k \frac{1}{|\mathbb{T}|} && \text{by uniformity} \\ &= \frac{1}{|\mathbb{T}|^k}. \end{aligned}$$

For the forward direction, suppose that \mathcal{H} is strongly k -universal. To prove uniformity, let $x \in \mathbb{U}$ and $y \in \mathbb{T}$. Let $x_2, \dots, x_k \in \mathbb{U} \setminus \{x\}$ be pairwise distinct. Then

$$\Pr_{h \in \mathcal{H}} (h(x) = y) = \sum_{y_2, \dots, y_k \in \mathbb{T}} \Pr_{h \in \mathcal{H}} \left(h(x) = y \wedge \bigwedge_{i=2}^k h(x_i) = y_i \right) = |\mathbb{T}|^{k-1} \cdot \frac{1}{|\mathbb{T}|^k} = \frac{1}{|\mathbb{T}|}.$$

To prove k -independence, let $x_1, \dots, x_k \in \mathbb{U}$ be distinct and $y_1, \dots, y_k \in \mathbb{T}$.
Then

$$\Pr_{h \in \mathcal{H}} \left(\bigwedge_{i=1}^k h(x_i) = y_i \right) = \frac{1}{|\mathbb{T}|^k} = \prod_{i=1}^k \Pr_{h \in \mathcal{H}} (h(x_i) = y_i),$$

where the second equality holds by uniformity. □

Construction of Strongly k -Universal Families

- ▶ We choose a prime power $q \geq N$ and let \mathbb{F}_q denote the field with q elements (unique up to isomorphism).
- ▶ We fix an arbitrary injective function $g_1 : \mathbb{U} \rightarrow \mathbb{F}_q$ and an arbitrary bijection $g_2 : \mathbb{F}_q \rightarrow \{0, \dots, q-1\}$.
- ▶ For $\mathbf{a} = (a_0, \dots, a_{k-1}) \in \mathbb{F}_q^k$, let $p_{\mathbf{a}} : \mathbb{F}_q \rightarrow \mathbb{F}_q$ be the polynomial function

$$p_{\mathbf{a}}(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{k-1} x^{k-1},$$

and let $f_{\mathbf{a}} : \mathbb{U} \rightarrow \{0, \dots, q-1\}$ be the function $g_2 \circ p_{\mathbf{a}} \circ g_1$.

Lemma 8.13

The family $\mathcal{H}_q^k := \{f_{\mathbf{a}} \mid \mathbf{a} \in \mathbb{F}_q^k\}$ of hash functions from \mathbb{U} to $\{0, \dots, q-1\}$ is strongly k -universal.

For a field \mathbb{F} , by $\mathbb{F}[X]$ we denote the set of all polynomials over \mathbb{F} in one variable X .

Fact (Lagrangian Interpolation)

Let \mathbb{F} be an arbitrary field. Let $k \geq 1$, and let $x_1, \dots, x_k \in \mathbb{F}$ be distinct and $y_1, \dots, y_k \in \mathbb{F}$. Then there is a unique polynomial $p(X) \in \mathbb{F}[X]$ of degree $k - 1$ such that

$$p(x_i) = y_i \quad \text{for all } i \in [k].$$

Proof sketch.

The coefficients a_0, \dots, a_{k-1} of the polynomial are solutions to the following system of linear equations

$$\underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{k-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^{k-1} \end{pmatrix}}_{=:A} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} \quad (*)$$

The matrix A of this system is a so-called **Van-der-Monde** matrix. It can be shown that the determinant of A is

$$\det(A) = \prod_{1 \leq i < j \leq n} (x_j - x_i),$$

which is nonzero if the x_i are mutually distinct. Thus A is nonsingular, and the system (\star) has a unique solution. □

Proof of the lemma.

The fact implies that for all distinct $x_1, \dots, x_k \in \mathbb{F}_q$ and all $y_1, \dots, y_k \in \mathbb{F}_q$ there is a unique $\mathbf{a} = (a_0, \dots, a_{k-1}) \in \mathbb{F}_q^k$ such that

$$p_{\mathbf{a}}(x_i) = y_i \quad \text{for all } i \in [k].$$

Thus

$$\Pr_{\mathbf{a} \in \mathbb{F}_q^k} (p_{\mathbf{a}}(x_1) = y_1 \wedge \dots \wedge p_{\mathbf{a}}(x_k) = y_k) = \frac{1}{q^k}.$$

This implies for all distinct $u_1, \dots, u_k \in \mathbb{U}$ and all $t_1, \dots, t_k \in \{0, \dots, q-1\}$,

$$\Pr_{h \in \mathcal{H}_q^k} (h(u_1) = t_1 \wedge \dots \wedge h(u_k) = t_k)$$

$$\begin{aligned}
&= \Pr_{a \in \mathbb{F}_q^k} (f_a(u_1) = t_1 \wedge \dots \wedge f_a(u_k) = t_k) \\
&= \Pr_{a \in \mathbb{F}_q^k} (p_a(g_1(u_1)) = g_2^{-1}(t_1) \wedge \dots \wedge p_a(g_1(u_k)) = g_2^{-1}(t_k)) \\
&= \frac{1}{q^k}.
\end{aligned}$$

□

Construction of Strongly k -Universal Families (cont'd)

It remains to construct a strongly k -universal family mapping \mathbb{U} to $\mathbb{T} := \{0, \dots, M - 1\}$ for an $M \ll N$.

- ▶ We choose a prime power $q \geq N$ and define the mappings $f_{\mathbf{a}} : \mathbb{U} \rightarrow \{0, \dots, q - 1\}$ for $\mathbf{a} \in \mathbb{F}_q^k$ as on the previous slide.
- ▶ We define functions $h_{\mathbf{a}} : \mathbb{U} \rightarrow \{0, 1, \dots, M - 1\}$ by

$$h_{\mathbf{a}}(x) := f_{\mathbf{a}}(x) \mod M.$$

- ▶ We let $\mathcal{H}_{q,M}^k := \{h_{\mathbf{a}} \mid \mathbf{a} \in \mathbb{F}_q^k\}$.

Theorem 8.14

If M divides q , then the family $\mathcal{H}_{q,M}^k$ is strongly k -universal.

Notes for Page 8.21

Proof of the Theorem.

Let $r := \frac{q}{M}$.

For all $y \in \{0, \dots, M-1\}$, let

$$F(y) := \{y + i \cdot M \mid 0 \leq i \leq r-1\}.$$

Then $|F(y)| = r$. Furthermore, for all $\mathbf{a} = (a_0, \dots, a_{k-1}) \in \mathbb{F}_q^k$,

$$F(y) = \{f_{\mathbf{a}}(x) \mid h_{\mathbf{a}}(x) = y\}.$$

Thus for distinct $x_1, \dots, x_k \in \mathbb{U}$ and $y_1, \dots, y_k \in \{0, \dots, M-1\}$,

$$\begin{aligned} \Pr_{h \in \mathcal{H}_{q,M}^k} \left(\bigwedge_{i=1}^k h(x_i) = y_i \right) &= \Pr_{\mathbf{a} \in \mathbb{F}_q^k} \left(\bigwedge_{i=1}^k h_{\mathbf{a}}(x_i) = y_i \right) \\ &= \Pr_{\mathbf{a} \in \mathbb{F}_q^k} \left(\bigvee_{z_1 \in F(y_1), \dots, z_k \in F(y_k)} \bigwedge_{i=1}^k f_{\mathbf{a}}(x_i) = z_i \right) \\ &= \sum_{z_1 \in F(y_1), \dots, z_k \in F(y_k)} \Pr_{\mathbf{a} \in \mathbb{F}_q^k} \left(\bigwedge_{i=1}^k f_{\mathbf{a}}(x_i) = z_i \right) \end{aligned}$$

$$\begin{aligned} &= r^k \cdot \frac{1}{q^k} \\ &= \frac{1}{M^k}. \end{aligned} \tag{Lemma}$$

□

Even if M does not divide q , the family $\mathcal{H}_{q,M}^k$ is close to strongly k -universal, as long as $M \ll q$.

Theorem 8.15

For all M , the family $\mathcal{H}_{q,M}^k$ satisfies the following two conditions.

Independence: For all distinct $x_1, \dots, x_k \in \mathbb{U}$ and all $y_1, \dots, y_k \in \{0, \dots, M-1\}$,

$$\Pr_{h \in \mathcal{H}} \left(\bigwedge_{i=1}^k h(x_i) = y_i \right) = \prod_{i=1}^k \Pr_{h \in \mathcal{H}} (h(x_i) = y_i).$$

Almost Uniformity: For all $x \in \mathbb{U}$ and all $y \in \{0, \dots, M-1\}$,

$$\left| \Pr_{h \in \mathcal{H}} (h(x) = y) - \frac{1}{M} \right| \leq \frac{1}{q}.$$

(Proof as exercise.)

8.3 Counting Distinct Elements

Counting Distinct Elements

Universe \mathbb{U} of size $|\mathbb{U}| = N$, data stream a_1, \dots, a_n of items from \mathbb{U}

Task

Count the number of distinct elements in a_1, \dots, a_n .

Observation 8.16

The problem can be solved using space $O(N)$ or space $O(n \log N)$.

Theorem 8.17

The problem cannot be solved (exactly) in space $< \min\{N, n\}$.

Proof of the observation.

To solve the problem in space $O(N)$, use a bit array of length N with one bit for every element of \mathbb{U} that is set to 1 once the element has appeared.

To solve the problem in space $O(n \log N)$, just store all elements that occur. □

Proof of the theorem.

Consider the state of the algorithm after the elements a_1, \dots, a_{n-1} have been read.

Claim

There must be at least as many states of the algorithm as there are nonempty subsets $S \subseteq \mathbb{U}$ of size at most $n - 1$.

Proof of the claim.

Suppose for contradiction that the claim is false. Then there are distinct subsets S_1, S_2 such that the algorithm assumes the same memory state after it has seen either of these sets.

Without loss of generality, we may assume that $|S_1| \leq |S_2|$.

Case 1: $|S_1| < |S_2|$.

Then if the last element a_n in the stream is from S_1 , the algorithm should answer $|S_1|$ if it has seen S_1 and either $|S_2|$ or $|S_2| + 1$ if it has seen S_2 (depending on whether $a_n \in S_2$) . But, having the same memory state after either of these sets, it can't know which answer to give.

Case 2: $|S_1| = |S_2|$.

Then if the last element a_n in the stream is from $S_1 \setminus S_2$, the algorithm should answer $|S_1|$ if it has seen S_1 and $|S_2| + 1$ if it has seen S_2 . Again, having the same memory state after either of these sets, it can't know which answer to give.

In both cases, the algorithm cannot give the correct answer, which leads to a contradiction and proves the claim. \square

The number of nonempty subsets $S \subseteq \mathbb{U}$ of size at most $n - 1$ is

$$s(N, n) := \sum_{i=1}^{n-1} \binom{N}{i}.$$

Thus the algorithm needs a memory space of at least $\lceil \log s(N, n) \rceil$ bits.

- ▶ If $n \geq N$ then $s(N, n) \geq 2^N - 2$, which implies $\lceil \log s(N, n) \rceil = N$ (assuming $N \geq 3$).
- ▶ If $N/2 < n \leq N - 1$, we have $s(N, n) \geq 2^N/2$ and thus $\log s(N, n) \geq N - 1 \geq n$.
- ▶ If $n \leq N/2$, we have

$$s(N, n) \geq \binom{N}{n-1} \geq \left(\frac{N}{n-1}\right)^{n-1} > 2^{n-1}$$

and thus $\lceil \log s(N, n) \rceil \geq n$.

Approximately Counting Distinct Elements

- ▶ As we assume N and n to be very large, the linear space lower bound for exact counting is prohibitive.
- ▶ But for many applications, it is sufficient to count the number of distinct elements in a data stream approximately.

Basic Idea: Size of a Random Set

Let $1 \leq m \leq N$, and let $\mathbb{U} := [N]$. Let $S \subseteq \mathbb{U}$ be a random m -element subset.

Idea

Use $\frac{N}{\min S}$ as an estimator for $|S|$.

Observation 8.18

$$E_{S \sim \binom{\mathbb{U}}{m}}(\min S) \approx \frac{N}{m}.$$

It follows that $E_{S \sim \binom{\mathbb{U}}{m}}\left(\frac{N}{\min S}\right) \approx m$.

Notes for Page 8.26

The assertion is not mathematically precise, and we neither attempt to make it precise here nor give a proof. We will just give an intuitive argument why it should be true. This will help to understand the algorithm described next.

Let us assume that $m \ll N$ (say, $m \leq \sqrt[3]{N}$). For every $i \in N$, let X_i be the indicator random variable for the event $i \in S$. Then

$$\Pr(X_i = 1) = \frac{m}{N}.$$

The X_i are not independent. For example, if $1 \notin S$ then it is more likely that $2 \in S$. However, by the assumption $m \ll N$, they are close to being independent.

For all $k \in [N]$ we have

$$\Pr(\min S = k) = \Pr(X_1 = 0, \dots, X_{k-1} = 0, X_k = 1)$$

That is, $\min S$ is approximately geometrically distributed with parameter $p = \frac{m}{N}$. The expected value of a random variable that is geometrically distributed with parameter p is $\frac{1}{p}$, which in our case is $\frac{N}{m}$.

Remark.

If you do not know what a geometric distribution is, try to figure it out by consulting a textbook on probability or statistics, for example, [\[Mitzenmacher and Upfal, 2005, Section 2.4\]](#), or [Wikipedia](#). This is something you should know.

Ideas of the Counting algorithm

- ▶ Use a random hash function $h : \mathbb{U} \rightarrow [M]$, for a sufficiently large M , to make the entries of the input stream look random.
- ▶ It will suffice to use a strongly 2-universal hash function.
- ▶ As a first approximation, we could compute the minimum hash value x appearing in the stream and return $\frac{M}{x}$ as our estimator for the number of distinct elements.
- ▶ It turns out that we get a more precise estimate if we compute the t -th smallest element, for some constant t depending on the approximation error, and return $\frac{tM}{x}$ as our estimator.

The Approximate Counting Algorithm

As usual, let $N := |\mathbb{U}|$. Let $M \geq 12N^2$, and let \mathcal{H} be a strongly 2-universal family of hash functions from \mathbb{U} to $[M]$. Let $0 <$

The algorithm COUNT maintains a sorted list x_1, \dots, x_t of the $t := \lceil \frac{24}{\epsilon^2} \rceil$ smallest hash values of stream elements and returns $\frac{tM}{x_t}$ as the estimator in the end.

Algorithm COUNT(ϵ)

1. $h \sim \mathcal{H}$
2. $t \leftarrow \lceil \frac{24}{\epsilon^2} \rceil$
3. $(x_1, \dots, x_t) \leftarrow (M+1, \dots, M+1)$
4. **while** not end of stream **do**
5. $a \leftarrow$ next stream element
6. INSERT($(x_1, \dots, x_t), h(a)$)
7. **if** $x_t \leq M$ **then**
8. **return** $\frac{tM}{x_t}$
9. **else**
10. $i \leftarrow \max\{j \leq t \mid x_j < M+1\}$
11. **return** i

INSERT($(x_1, \dots, x_t), y$)

1. $i = 1$
2. **while** $y > x_i$ **do**
3. $i \leftarrow i + 1$
4. **if** $i \leq t$ and $y < x_i$ **then**
5. **for** $j = i + 1$ to t **do**
6. $x_j \leftarrow x_{j-1}$
7. $x_i \leftarrow y$
8. **return** (x_1, \dots, x_t)

Analysis of the Counting Algorithm

Space

Algorithm COUNT(ε) needs memory space

$$O(t \log M) = O\left(\frac{1}{\varepsilon^2} \log N\right).$$

Approximation Guarantee

Let m be the number of distinct elements in the input stream, and let m^* be the estimator returned by COUNT(ε). Then

$$\Pr\left((1 - \varepsilon)m \leq m^* \leq (1 + \varepsilon)m\right) \geq \frac{3}{4}.$$

Proof of the approximation guarantee.

Since \mathcal{H} is strongly 2-universal, for all $u, u' \in \mathbb{U}$ we have

$$\Pr(h(u) = h(u')) \leq \frac{1}{M}.$$

Hence

$$\Pr(h \text{ is not injective}) \leq \frac{N^2}{M} \leq \frac{1}{12}.$$

Suppose that the input stream is a_1, \dots, a_n , and let b_1, \dots, b_m be pairwise distinct such that $\{b_1, \dots, b_m\} = \{a_1, \dots, a_n\}$. Moreover, let

$$m' := |\{h(b_j) \mid j \in [m]\}|$$

Then $m' \leq m$, and $m' = m$ if h is injective.

Assume first that $m' < t$. Then at the end of the stream, $x_1, \dots, x_{m'}$ will be a list of the distinct hash values of elements appearing in the stream, and

$x_{m'+1} = x_{m'+2} = \dots = x_t = M + 1$. In this case, the algorithm COUNT(ε) will return $m^* = m'$. Thus

$$\Pr(m^* = m) \geq \Pr(h \text{ is injective}) \geq \frac{11}{12},$$

which implies the desired bound.

In the following, assume that $m' \geq t$. Then

$$m^* = \frac{tM}{x_t} \tag{*}$$

We shall prove that

$$\Pr(m^* > (1 + \varepsilon)m) \leq \frac{1}{12}, \tag{**}$$

$$\Pr(m^* < (1 - \varepsilon)m) \leq \frac{2}{12}. \tag{\clubsuit}$$

This will imply the desired bound

$$\Pr((1 - \varepsilon)m \leq m^* \leq (1 + \varepsilon)m) \geq \frac{3}{4}.$$

We first prove (**). Suppose that $m^* > (1 + \varepsilon)m$. Then, by (*), $x_t < \frac{tM}{(1+\varepsilon)m}$. Since $x_1 < x_2 < \dots < x_t$, this implies

$$x_i < \frac{tM}{(1+\varepsilon)m} \quad \text{for all } i \leq t. \quad (\spadesuit)$$

For every $j \in [m]$, let X_j be the indicator random variable for the event $h(b_j) < \frac{tM}{(1+\varepsilon)m}$. Since our family \mathcal{H} of hash function is strongly universal, the X_j are pairwise independent, and we have

$$\mathbb{E}(X_j) = \Pr(X_j = 1) = \frac{\left\lceil \frac{tM}{(1+\varepsilon)m} \right\rceil - 1}{M} \leq \frac{\frac{tM}{(1+\varepsilon)m}}{M} = \frac{t}{(1+\varepsilon)m}.$$

This implies

$$\text{Var}(X_j) = \mathbb{E}(X_j^2) - \mathbb{E}(X_j)^2 = \mathbb{E}(X_j) - \mathbb{E}(X_j)^2 \leq \mathbb{E}(X_j) \leq \frac{t}{(1+\varepsilon)m}.$$

Let $X = \sum_{j=1}^m X_j$. Then

$$\mathbb{E}(X) = \sum_{j=1}^m \mathbb{E}(X_j) \leq \frac{t}{1+\varepsilon}.$$

Since the X_j are pairwise independent, by Lemma 2.1 we have

$$\text{Var}(X) = \sum_{j=1}^m \text{Var}(X_j) \leq \frac{t}{1+\varepsilon}.$$

By Chebychev's inequality, for all $c > 0$ we have

$$\Pr(|X - \mathbb{E}(X)| \geq ct) \leq \frac{\text{Var}(X)}{c^2 t^2} \leq \frac{1}{(1+\varepsilon)c^2 t}. \quad (\heartsuit)$$

Since $\{x_1, \dots, x_t\} \subseteq \{h(b_j) \mid j \in [m]\}$, () implies that at least t of the X_j evaluate to 1 and thus $X \geq t$. Hence

$$X - \mathbb{E}(X) \geq t - \frac{t}{1+\varepsilon} = \frac{\varepsilon}{1+\varepsilon} t$$

Let $c := \frac{\varepsilon}{1+\varepsilon}$. We have proved that if $m^* > (1+\varepsilon)m$ then $X - E(X) \geq ct$. By () it follows that

$$\Pr(|X - \mathbb{E}(X)| \geq ct) \leq \frac{1}{(1+\varepsilon)c^2 t} = \frac{(1+\varepsilon)}{\varepsilon^2 t} \leq \frac{2}{\varepsilon^2 \frac{24}{\varepsilon^2}} = \frac{1}{12},$$

where the last inequality holds because $\varepsilon \leq 1$ and $t \geq \frac{24}{\varepsilon^2}$. Thus $\Pr(m^* > (1 + \varepsilon)m) \leq \frac{1}{12}$, which proves (**).

It remains to prove (♣). Suppose that $m^* < (1 - \varepsilon)m$. This implies that $x_t > \frac{tM}{(1-\varepsilon)m}$. Thus

$$S := \left| \left\{ h(b_j) \mid j \in [m] \text{ with } h(b_j) \leq \frac{tM}{(1-\varepsilon)m} \right\} \right| < t.$$

For every $j \in [m]$, let Y_j be the indicator random variable for the event $h(b_j) \leq \frac{tM}{(1-\varepsilon)m}$, and let $Y := \sum_{j=1}^m Y_j$. Note that if h is injective we have $S = Y$, but in general we only have $S \leq Y$.

The Y_j are pairwise independent and we have

$$\mathbb{E}(Y_j) = \Pr(Y_j = 1) = \frac{\left\lfloor \frac{tM}{(1-\varepsilon)m} \right\rfloor}{M}.$$

Hence

$$\mathbb{E}(Y_j) \leq \frac{\frac{tM}{(1-\varepsilon)m}}{M} = \frac{t}{(1 - \varepsilon)m},$$

$$\mathsf{E}(Y_j) \geq \frac{\frac{tM}{(1-\varepsilon)m} - 1}{M} = \frac{t}{(1-\varepsilon)m} - \frac{1}{M}.$$

Since $m \leq M$, it follows that

$$\frac{t}{(1-\varepsilon)} - 1 \leq E(Y) \leq \frac{t}{1-\varepsilon}.$$

Furthermore, $\text{Var}(Y_j) \leq E(Y_j) \leq \frac{t}{(1-\varepsilon)m}$ and thus

$$\text{Var}(Y) = \sum_{j=1}^m \text{Var}(Y_j) \leq \frac{t}{1-\varepsilon}.$$

By Chebychev's inequality, for all $c > 0$ we have

$$\Pr(|Y - \mathsf{E}(Y)| \geq ct) \leq \frac{\text{Var}(Y)}{c^2 t^2} \leq \frac{1}{(1-\varepsilon)c^2 t}. \quad (\diamond)$$

Observe that if $Y < t$ then

$$E(Y) - Y \geq \left(\frac{t}{1-\varepsilon} - 1 \right) - (t-1) = \frac{\varepsilon}{1-\varepsilon} t.$$

Thus by (\diamond) with $c = \frac{\varepsilon}{1-\varepsilon}$,

$$\Pr(Y < t) \leq \Pr\left(E(Y) - Y \geq \frac{\varepsilon}{1-\varepsilon}t\right) \leq \frac{1-\varepsilon}{\varepsilon^2 t} \leq \frac{1}{12}.$$

Recall that if $m^* < (1 - \varepsilon)m$ we have $S < t$. Moreover, if h is injective, we have $Y = S$. Putting everything together, we get

$$\begin{aligned}\Pr(m^* < (1 - \varepsilon)m) \\ &\leq \Pr(S < t) \\ &= \Pr(S < t \text{ and } h \text{ is injective}) + \Pr(S < t \text{ and } h \text{ is not injective}) \\ &= \Pr(Y < t \text{ and } h \text{ is injective}) + \Pr(S < t \text{ and } h \text{ is not injective}) \\ &\leq \Pr(Y < t) + \Pr(h \text{ is not injective}) \\ &\leq \frac{2}{12}.\end{aligned}$$

This proves (\clubsuit) .

□

The Median Trick

Recall that the **median** of numbers $b_1 \leq b_2 \leq \dots \leq b_k$ is $b_{\lfloor \frac{k+1}{2} \rfloor}$.

Median Trick

For some $k \geq 1$,

- ▶ Run $2k - 1$ copies of $\text{COUNT}(\varepsilon)$ in parallel with hash functions h_1, \dots, h_{2k-1} drawn independently from \mathcal{H} .
- ▶ Let $m^{(1)}, \dots, m^{(2k-1)}$ be the resulting estimators for the number d of distinct elements in the input stream.
Return the median m^* of $m^{(1)}, \dots, m^{(2k-1)}$.

Call this version of the algorithm **MCOUNT(ε, k)**.

Key Observation

Let $\ell \in \mathbb{N}$. If $m^* \geq \ell$, then $m^{(i)} \geq \ell$ for at least k indices $i \in [2k - 1]$.
If $m^* \leq \ell$, then $m^{(i)} \leq \ell$ for at least k indices $i \in [2k - 1]$.

Notes for Page 8.30

If k is even, the median is often defined as the arithmetic mean of the two middle values, that is,

$$\frac{b_{k/2} + b_{k/2+1}}{2}.$$

This is irrelevant for us, because we are only going to use the median of an odd number of numbers.

Space

Algorithm MCOUNT(ε, k) needs memory space

$$O\left(\frac{k \log N}{\varepsilon^2}\right).$$

Approximation Guarantee

For every $\delta > 0$ there exists a $k = O(\ln(1/\delta))$ such that the estimator m^* returned by MCOUNT(ε, k) satisfies

$$\Pr\left((1 - \varepsilon)m \leq m^* \leq (1 + \varepsilon)m\right) \geq 1 - \delta.$$

As before, m denotes the number of distinct elements in the input stream.

Proof of the approximation guarantee.

Let $m^{(1)}, \dots, m^{(2k-1)}$ be the estimators produced by the $(2k - 1)$ copies of COUNT(ε) run in parallel. Then m^* is the median of $m^{(1)}, \dots, m^{(2k-1)}$. For $j \in [2k - 1]$, let X_j be the indicator random variable for the event $m^{(j)} > (1 + \varepsilon)m$. Then

$$\Pr(X_j = 1) \leq \frac{1}{4}$$

Let $X := \sum_{j=1}^{2k-1} X_j$. Note that

$$\Pr(m^* > (1 + \varepsilon)m) = \Pr(X \geq k).$$

We have

$$\mathbb{E}(X) \leq \frac{1}{4}(2k - 1) = \frac{1}{2}k - \frac{1}{4}.$$

and thus

$$k - \mathbb{E}(X) \geq \frac{1}{2}k + \frac{1}{4} \geq \frac{1}{4}(2k - 1).$$

By the Hoeffding Bound (Theorem 2.8) we have

$$\begin{aligned}\Pr(m^* > (1 + \varepsilon)m) &= \Pr(X \geq k) \\ &\leq \Pr\left(X \geq \mathbb{E}(X) + \frac{1}{4}(2k - 1)\right) \\ &\leq \exp\left(-\frac{2k - 1}{8}\right).\end{aligned}$$

For $k \geq 4 \ln\left(\frac{2}{\delta}\right) + \frac{1}{2}$, this yields

$$\Pr(m^* > (1 + \varepsilon)m) \leq \frac{\delta}{2}.$$

By a similar argument, we can show that for $k \geq 4 \ln\left(\frac{2}{\delta}\right) + \frac{1}{2}$ we have

$$\Pr(m^* < (1 - \varepsilon)m) \leq \frac{\delta}{2}.$$

This yields the desired bound. □

8.4 Frequency Moments

Let $\mathbf{a} = a_1, \dots, a_n$ be a data stream of elements from \mathbb{U} .

- ▶ Let $u \in \mathbb{U}$. The **frequency** of u in \mathbf{a} is

$$f_u(\mathbf{a}) := |\{i \in [n] \mid a_i = u\}|.$$

- ▶ Let $p \geq 0$ be a nonnegative real. The **p th frequency moment** of \mathbf{a} is

$$F_p(\mathbf{a}) := \sum_{u \in \mathbb{U}} (f_u(\mathbf{a}))^p.$$

For $p = 0$ we restrict the sum to the $u \in \mathbb{U}$ with $f_u(\mathbf{a}) > 0$.

- ▶ If the stream \mathbf{a} is clear from the context, we just write f_u and F_p instead of $f_u(\mathbf{a})$ and $F_p(\mathbf{a})$.

Interpretation

- ▶ F_0 is the number of distinct elements in \mathbf{a} .
- ▶ F_1 is the length n of the stream \mathbf{a} .
- ▶ $\sqrt{F_2} = \|\mathbf{f}\|$ is the Euclidean norm of the vector $\mathbf{f} = (f_u)_{u \in \mathbb{U}}$.
- ▶ More generally, $\sqrt[p]{F_p} =: \|\mathbf{f}\|_p$ is the ℓ_p -norm of \mathbf{f} .
- ▶ $\sqrt[p]{F_p}$ converges to the maximum frequency of any element of \mathbb{U} :

$$\sqrt[p]{F_p} = \|\mathbf{f}\|_p \xrightarrow[p \rightarrow \infty]{} \|\mathbf{f}\|_{\max} = \max\{f_u \mid u \in \mathbb{U}\}.$$

The second frequency moment can be used to compute the **average variance** of the stream, that is, the average squared distance from the expected frequency.

For this, we assume the elements of the stream to be chosen randomly from the universe, and we assume that on average each element appears the same number of times. That is,

$$\mathbb{E}(f_u) = \frac{n}{N}$$

for every $u \in \mathbb{U}$.

Thus the average variance is

$$\begin{aligned}\frac{1}{N} \sum_{u \in \mathbb{U}} \mathbb{E} (f_u - \mathbb{E}(f_u))^2 &= \frac{1}{N} \sum_{u \in \mathbb{U}} \left(\mathbb{E}(f_u^2) - \mathbb{E}(f_u)^2 \right) \\ &= \frac{1}{N} \sum_{u \in \mathbb{U}} \mathbb{E}(f_u^2) - \frac{1}{N} \cdot N \cdot \frac{n^2}{N^2} \\ &= \frac{1}{N} F_2 - \frac{n^2}{N^2}.\end{aligned}$$

An Estimator for F_k

Let $k \geq 2$ be an integer. Let $\mathbf{a} = a_1, \dots, a_n$ be the input stream.

Estimator A_k

- ▶ Pick an index $i \in [n]$ uniformly at random.
- ▶ Let $r := |\{j \geq i \mid a_j = a_i\}|$ (the number of occurrences of a_i in the “rest of the stream” starting at position i).
- ▶ Let $A_k := n(r^k - (r-1)^k)$.

Theorem 8.19

$$\mathbb{E}(A_k) = F_k.$$

Proof of the theorem.

For every $u \in \mathbb{U}$, let

$$I_u := \{i \in [n] \mid a_i = u\}.$$

Note that $|I_u| = f_u$. Suppose that $I_u = \{i_{u1}, \dots, i_{uf_u}\}$.

We think of the uniform choice of the index $i \in [n]$ as a 2-stage process:

- (1) Pick $u \in \mathbb{U}$ with probability $\frac{f_u}{n}$.
- (2) Pick $j \in [f_u]$ uniformly at random and return i_{uj} .

Let C_u be the event that u is chosen in the first step and D_j the event that j is chosen in the second step. Then

$$\Pr(C_u) = \frac{f_u}{n},$$

$$\Pr(D_j | C_u) = \frac{1}{f_u}.$$

Moreover, if C_u and D_j occur then $r = f_u - j + 1$, and thus the value of A_k is

$$A(u, j) := n((f_u - j + 1)^k - (f_u - j)^k).$$

Note that

$$\begin{aligned}\sum_{j=1}^{f_u} A(u, j) &= n \left(\sum_{j=1}^{f_u} (f_u - j + 1)^k - \sum_{j=1}^{f_u} (f_u - j)^k \right) \\ &= n \left(\sum_{j=0}^{f_u-1} (f_u - j)^k - \sum_{j=1}^{f_u} (f_u - j)^k \right) \\ &= n(f_u^k - 0^k) = nf_u^k.\end{aligned}$$

Thus

$$\begin{aligned}\mathsf{E}(A_k) &= \sum_{u \in \mathbb{U}} \sum_{j \in [f_u]} \Pr(C_u \wedge D_j) A(u, j) \\&= \sum_{u \in \mathbb{U}} \Pr(C_u) \sum_{j=1}^{f_u} \Pr(D_j \mid C_u) A(u, j) \\&= \sum_{u \in \mathbb{U}} \frac{f_u}{n} \cdot \frac{1}{f_u} \sum_{j=1}^{f_u} A(u, j) \\&= \sum_{u \in \mathbb{U}} f_u^k \\&= F_k.\end{aligned}$$

□

Implementation

To sample an index i uniformly at random, we use the “reservoir sampling” trick.

Algorithm AMS-ESTIMATOR

1. $i = 0$
2. **while** not end of stream **do**
3. $i \leftarrow i + 1$
4. **with probability** $1/i$ **do**
5. $a \leftarrow a_i$
6. $r \leftarrow 0$
7. **if** $a_i = a$ **then**
8. $r \leftarrow r + 1$
9. **return** $i(r^k - (r - 1)^k)$



N. Alon



Y. Matias



M. Szegedy

Remark 8.20

The “AMS” refers to Noga Alon, Yossi Matias, and Mario Szegedy, who proposed this estimator.

- ▶ Unfortunately, the variance of the estimator A_k is very high.
- ▶ This means that we cannot get a good upper bound on the probability p that A_k is close to F_k via Chebychev's inequality.
- ▶ In fact, the probability p is below $1/2$, so the median trick for improving the error probability does not work.
- ▶ There is a more complicated way of improving the error probability ("median-of-means trick"), but it requires space $O(n^{1-1/k} \text{polylog } n)$.
- ▶ This is sublinear, but far from the desired $\text{polylog}(n + N)$.

A Better Estimator for F_2

Let \mathcal{H} be a strongly 4-universal family of hash functions from \mathbb{U} to $\{-1, 1\}$.

Algorithm TUG-OF-WAR

1. draw h uniformly at random from \mathcal{H}
2. $x \leftarrow 0$
3. **while** not end of stream **do**
4. $a \leftarrow$ next element from stream
5. $x \leftarrow x + h(a)$
6. **return** x^2

Lemma 8.21

Let B be the estimator returned by the algorithm Tug-of-War. Then $E(B) = F_2$ and $\text{Var}(B) \leq 2F_2^2$.

Proof of the lemma.

Let X be the value of the variable x after the stream has been processed. Then $B = X^2$. For $u \in \mathbb{U}$, let $Y_u = h(u)$. Then

$$X = \sum_{u \in \mathbb{U}} f_u Y_u.$$

Thus

$$\mathbb{E}(X^2) = \mathbb{E} \left(\sum_{u,v \in \mathbb{U}} f_u f_v Y_u Y_v \right) = \sum_{u \in \mathbb{U}} f_u^2 \mathbb{E}(Y_u^2) + \sum_{\substack{u,v \in \mathbb{U} \\ u \neq v}} f_u f_v \mathbb{E}(Y_u Y_v)$$

We have $Y_u^2 = 1$ and thus $\mathbb{E}(Y_u^2) = 1$. Furthermore, since \mathcal{H} is strongly 4-universal, we have $\Pr(Y_u = 1) = \Pr(Y_u = -1) = \frac{1}{2}$ and thus $\mathbb{E}(Y_u) = 0$. Moreover, for $u \neq v$ the variables Y_u and Y_v are independent. Hence

$$\mathbb{E}(Y_u Y_v) = \mathbb{E}(Y_u) \mathbb{E}(Y_v) = 0.$$

It follows that

$$\mathbb{E}(X^2) = \sum_{u \in \mathbb{U}} f_u^2 = F_2.$$

For the variance, we first note that

$$\text{Var}(X^2) = \mathbb{E}(X^4) - \mathbb{E}(X^2)^2 = \mathbb{E}(X^4) - F_2^2.$$

We have

$$\begin{aligned}\mathbb{E}(X^4) &= \mathbb{E} \left(\sum_{u_1, u_2, u_3, u_4 \in \mathbb{U}} f_{u_1} f_{u_2} f_{u_3} f_{u_4} Y_{u_1} Y_{u_2} Y_{u_3} Y_{u_4} \right) \\ &= \sum_{u_1, u_2, u_3, u_4 \in \mathbb{U}} f_{u_1} f_{u_2} f_{u_3} f_{u_4} \mathbb{E}(Y_{u_1} Y_{u_2} Y_{u_3} Y_{u_4}).\end{aligned}$$

Since \mathcal{H} is strongly 4-universal, the distinct variables among $Y_{u_1}, Y_{u_2}, Y_{u_3}, Y_{u_4}$ are independent, and we have

$$\mathbb{E}(Y_{u_1} Y_{u_2} Y_{u_3} Y_{u_4}) = \begin{cases} 0 & \text{if there is an } i \text{ such that } u_i \neq u_j \text{ for all } j \neq i, \\ 1 & \text{otherwise.} \end{cases}$$

To see this, note that if, for example, $u_1 \notin \{u_2, u_3, u_4\}$ then Y_{u_1} is independent from $Y_{u_2} Y_{u_3} Y_{u_4}$ and we have

$$E(Y_{u_1} Y_{u_2} Y_{u_3} Y_{u_4}) = \underbrace{E(Y_{u_1})}_{=0} \cdot E(Y_{u_2} Y_{u_3} Y_{u_4}) = 0.$$

The “otherwise” case is met if either $(u_1, u_2, u_3, u_4) = (u, u, u, u)$ for some $u \in \mathbb{U}$ or if there are distinct $u, v \in \mathbb{U}$ such that (u_1, u_2, u_3, u_4) is a permutation of (u, u, v, v) . There are six distinct permutations: $(u, u, v, v), (u, v, u, v), (u, v, v, u), (v, u, u, v), (v, u, v, u), (v, v, u, u)$. Thus

$$E(X^4) = \sum_{u \in \mathbb{U}} f_u^4 + 6 \sum_{\{u, v\} \in \binom{\mathbb{U}}{2}} f_u^2 f_v^2 = F_4 + 6 \sum_{\{u, v\} \in \binom{\mathbb{U}}{2}} f_u^2 f_v^2$$

Note that

$$F_2^2 = \left(\sum_{u \in \mathbb{U}} f_u^2 \right)^2 = \sum_{u, v \in \mathbb{U}} f_u^2 f_v^2 = \sum_{u \in \mathbb{U}} f_u^4 + 2 \sum_{\{u, v\} \in \binom{\mathbb{U}}{2}} f_u^2 f_v^2 = F_4 + 2 \sum_{\{u, v\} \in \binom{\mathbb{U}}{2}} f_u^2 f_v^2.$$

Thus

$$2 \sum_{\{u, v\} \in \binom{\mathbb{U}}{2}} f_u^2 f_v^2 = F_2^2 - F_4.$$

It follows that

$$\mathbb{E}(X^4) = 3F_2^2 - 2F_4$$

and thus

$$\text{Var}(X^2) = \mathbb{E}(X^4) - F_2^2 = 2F_2^2 - 2F_4 \leq 2F_2^2.$$



Lemma 8.22

Let X_1, \dots, X_k be pairwise independent identically distributed random variables with expected value $\mu > 0$ and variance σ^2 , and let $X = \frac{1}{k} \sum_{i=1}^k X_i$ (the average of the X_i). Let $\varepsilon, \delta > 0$ and suppose that

$$k \geq \frac{\sigma^2}{\varepsilon^2 \delta \mu^2}.$$

Then

$$\Pr(|X - \mu| < \varepsilon\mu) > 1 - \delta.$$

Proof of the lemma.

The key observation is that

$$\begin{aligned}
 \text{Var}(X) &= E(X^2) - \mu^2 \\
 &= \frac{1}{k^2} \sum_{i,j \in [k]} (E(X_i X_j) - \mu^2) \\
 &= \frac{1}{k^2} \sum_{i \in [k]} (E(X_i^2) - \mu^2) \\
 &= \frac{\sigma^2}{k}.
 \end{aligned} \tag{*}$$

Equality (*) holds because the X_i are pairwise independent and thus $E(X_i X_j) = E(X_i)E(X_j) = \mu^2$ for $i \neq j$.

Then by Chebychev's inequality we get

$$\Pr(|X - \mu| \geq \varepsilon\mu) \leq \frac{\text{Var}(X)}{\varepsilon^2 \mu^2} = \frac{\sigma^2}{k\varepsilon^2 \mu^2} \leq \delta.$$



Averaging the Tug-of-War Estimator

Algorithm AVG-ToW(k)

1. draw h_1, \dots, h_k independently from \mathcal{H}
2. for $i = 1, \dots, k$ do
3. $x_i \leftarrow 0$
4. while not end of stream do
5. $a \leftarrow$ next element from stream
6. for $i = 1, \dots, k$ do
7. $x_i \leftarrow x_i + h_i(a)$
8. return $\frac{1}{k} \sum_{i=1}^k x_i^2$

Theorem 8.23

Let $\varepsilon, \delta > 0$ and $k = \lceil \frac{2}{\varepsilon^2 \delta} \rceil$. Let B be the estimator returned by the algorithm AVG-ToW(k). Then $E(B) = F_2$ and

$$\Pr(|B - F_2| < \varepsilon F_2) > 1 - \delta.$$

Proof of the theorem.

Let B_i be the value x_i^2 at the end of the run of the algorithm. Then the B_i are independent and identically distributed with

$$\begin{aligned}\mu &:= \mathbb{E}(B_i) = F_2, \\ \sigma^2 &:= \text{Var}(B_i) \leq 2F_2^2.\end{aligned}$$

Furthermore, we have $B = \frac{1}{k} \sum_{i=1}^k B_i$ and

$$k \geq \frac{2}{\varepsilon^2 \delta} = \frac{2F_2^2}{\varepsilon^2 \delta F_2^2} \geq \frac{\sigma^2}{\varepsilon^2 \delta \mu^2}.$$

Now the theorem follows from the “averaging lemma”. □

8.5 Sketching

Sketching a Vector

Setting

- ▶ Data is stored in a (long) vector, which we want to query.
- ▶ The vector changes over time according to a sequence of updates, which arrive as a data stream.
- ▶ The vector as well as the stream of updates are too large to be stored in main memory.

Goal

A data structure that maintains a summary, or *sketch*, of the vector and

- ▶ enables us to answer queries approximately;
- ▶ allows for efficient updates;
- ▶ is space-efficient (and thus can be stored in main memory).

Formal Setting: the Turnstile Model

Turnstile Data Stream Model

- Universe \mathbb{U} of size N .
- Stream of updates $(a_1, c_1), \dots, (a_n, c_n)$, where $a_i \in \mathbb{U}$ and $c_i \in \mathbb{Z}$.
- Implicitly, for $0 \leq i \leq n$ we have a data vector

$$\mathbf{d}(i) = (d_u(i))_{u \in \mathbb{U}} \in \mathbb{Z}^{\mathbb{U}}$$

defined by

$$d_u(0) := 0 \quad \text{for all } u \in \mathbb{U},$$
$$d_u(i+1) := \begin{cases} d_u(i) + c_i & \text{if } u = a_i, \\ d_u(i) & \text{if } u \neq a_i, \end{cases} \quad \text{for all } u \in \mathbb{U}, i \in [n].$$

We let $\mathbf{d} := \mathbf{d}(n)$.

- We let $M := \sum_{i=1}^n |c_i|$.
- Our goal is to design data structures and algorithms that need memory space $\text{polylog}(N + M)$.

- ▶ In the **strict turnstile model** we assume that all entries of the data vector are nonnegative at any time, that is, $d_u(i) \geq 0$ for all $u \in \mathbb{U}$ and $i \in [n]$.
- ▶ In the **cash register model**, all updates are required to be positive, that is, $c_i > 0$ for all $i \in [n]$.
- ▶ The “normal” data stream model can be viewed as a special case of the cash register model if we interpret a stream element a_i as the update $(a_i, 1)$.

Then the data vector \mathbf{d} is just the frequency vector \mathbf{f} .

A Simple Sketch

Let $k \geq 1$, and let \mathcal{H} be a universal family of hash functions from \mathbb{U} to $[k]$.

Algorithm SIMPLE SKETCH(k)

1. draw h from \mathcal{H}
2. for $i = 1, \dots, k$ do
3. $S[i] := 0$
4. while not end of stream do
5. $(a, c) \leftarrow$ next update
6. $S[h(a)] \leftarrow S[h(a)] + c$
7. return S

Estimating Data Values with Simple Sketch

To estimate a data value d_u , we use the value $d_u^* := S[h(u)]$, where S is the array returned by Simple Sketch(k).

We will bound the error in terms of the ℓ_1 -norm of the data vector:

$$\|\mathbf{d}\|_1 := \sum_{u \in \mathbb{U}} d_u.$$

Lemma 8.24

Let $\varepsilon > 0$ such that $k \geq \frac{2}{\varepsilon}$. Then in the strict turnstile model, the following holds for the estimator d_u^* :

- (i) $d_u \leq d_u^*$.
- (ii) $E(d_u^* - d_u) \leq \frac{\varepsilon}{2} \|\mathbf{d}\|_1$ and $\Pr(d_u^* - d_u \geq \varepsilon \|\mathbf{d}\|_1) \leq \frac{1}{2}$.

Proof of the lemma.

We have

$$d_u^* = d_u + \sum_{\substack{v \in \mathbb{U} \setminus \{u\} \\ h(u) = h(v)}} d_v$$

Since we are in the strict turnstile model, we have $d_v \geq 0$ for all v and thus $d_u^* \geq d_u$. This proves (i).

To prove (ii), let X_v be the indicator variable of the event $h(u) = h(v)$. Then for $v \neq u$,

$$\Pr(X_v) = \frac{1}{k},$$

because the family \mathcal{H} is universal. Let

$$X := \sum_{v \in \mathbb{U} \setminus \{u\}} d_v X_v = d_u^* - d_u.$$

We have

$$\mathbb{E}(X) = \frac{1}{k} \sum_{v \in \mathbb{U} \setminus \{u\}} d_v \leq \frac{1}{k} \|\mathbf{d}\|_1 \leq \frac{\epsilon}{2} \|\mathbf{d}\|_1.$$

By Markov's inequality,

$$\Pr(X \geq \epsilon \|\mathbf{d}\|_1) \leq \Pr(X \geq 2 \mathsf{E}(X)) \leq \frac{1}{2}.$$



Reducing the Error Probability

Let $k, \ell \geq 1$, and let \mathcal{H} be a universal family of hash functions from \mathbb{U} to $\{0, \dots, k - 1\}$.

Algorithm COUNT MIN SKETCH(k, ℓ)

1. draw h_1, \dots, h_ℓ independently from \mathcal{H}
2. for $i = 1, \dots, k$ do
3. for $j = 1, \dots, \ell$ do
4. $S[i, j] \leftarrow 0$
5. while not end of stream do
6. $(a, c) \leftarrow$ next update
7. for $j = 1, \dots, \ell$ do
8. $S[h_j(a), j] \leftarrow S[h_j(a), j] + c$
9. return S

We call the array S returned by COUNT MIN SKETCH(k, ℓ) a **CM sketch** with parameters k, ℓ .

Estimating Data Values with Count Min Sketch

To estimate a data value d_u , we use the value

$$d_u^* := \min_{j \in [\ell]} S[h_j(u), j],$$

where S is a CM sketch.

Theorem 8.25

Let $\varepsilon, \delta > 0$ such that $k \geq \frac{2}{\varepsilon}$ and $\ell \geq \log \frac{1}{\delta}$. Then in the strict turnstile model, the following holds for the estimator d_u^* :

- (i) $d_u \leq d_u^*$.
- (ii) $\Pr(d_u^* - d_u \geq \varepsilon \|\mathbf{d}\|_1) \leq \delta$.

Proof of the Theorem.

Let $d_u^j := S[h_j(u), j]$ and $X^j := d_u^j - d_u$. Then

$$d_u^* - d_u = \min_{j \in [\ell]} X^j.$$

By the lemma, we have $X^j \geq 0$, which implies (i), and

$$E(X^j) \leq \frac{\varepsilon}{2} \|\mathbf{d}\|_1.$$

Note that the random variables X^1, \dots, X^ℓ are independent. Thus

$$\begin{aligned} \Pr(d_u^* - d_u \geq \varepsilon \|\mathbf{d}\|_1) &= \Pr\left(\bigwedge_{i=1}^{\ell} X^i \geq \varepsilon \|\mathbf{d}\|_1\right) \\ &= \prod_{i=1}^{\ell} \Pr(X^i \geq \varepsilon \|\mathbf{d}\|_1) \quad \text{by independence} \\ &\leq \frac{1}{2^\ell} \leq \delta. \end{aligned}$$



An element $u \in \mathbb{U}$ is a **heavy hitter** with **threshold** $\tau > 0$ for a data vector \mathbf{d} if

$$d_u \geq \tau \|\mathbf{d}\|_1.$$

Observation 8.26

There are at most $\frac{1}{\tau}$ heavy hitters with threshold τ .

We will use the count min sketch to approximately compute the heavy hitters of a data stream. For simplicity, we focus on the **cash register model** (all updates are positive).

Heavy Hitters with CM Sketch

Let $\tau > 0$ and $k, \ell \geq 1$, and let \mathcal{H} be a universal family of hash functions from \mathbb{U} to $\{0, \dots, k - 1\}$.

Algorithm CM HEAVY HITTERS(k, ℓ, τ)

- ▶ compute a CM sketch S with parameters k, ℓ
- ▶ maintain $\|\mathbf{d}\|_1$ during the computation
- ▶ during the computation, maintain a set H of elements $u \in \mathbb{U}$ whose estimated value $d_u^* := \min_j S[h_j(u), j]$ is at least $\tau \|\mathbf{d}\|_1$
- ▶ after each update (or whenever H gets too large), remove those elements u whose value d_u^* has dropped below $\tau \|\mathbf{d}\|_1$ from H
- ▶ return all $u \in H$ with $d_u^* \geq \tau \|\mathbf{d}\|_1$

Remark 8.27

An appropriate data structure for storing the set H is a priority queue.

Theorem 8.28

We assume the cash register model.

Let $\varepsilon, \delta, \tau > 0$ and $k \geq \frac{2}{\varepsilon}$ and $\ell \geq \log \frac{n}{\delta}$.

Then the algorithm CM Heavy Hitters(k, ℓ, τ) returns

- (i) all elements u such that $d_u \geq \tau \|\mathbf{d}\|_1$,
- (ii) with probability at least $1 - \delta$ no elements u such that $d_u \leq (\tau - \varepsilon) \|\mathbf{d}\|_1$.

Proof sketch.

The algorithm is correct, because in the cash register model, $\frac{d_u}{\|d\|_1}$ can only increase at an update (u, c) . So we only need to include elements that are currently updated into the set H .

By the previous theorem, for all $u \in \mathbb{U}$ we have $d_u^* \geq d_u$. This implies (i).

The probability that an estimate d_u^* deviates from d_u by more than ε is at most $\frac{\delta}{n}$. Hence by the union bound, the probability that this happens for at least one of the n elements seen during the execution of the algorithm is at most δ . This implies (ii). □

[Leskovec et al., 2014, Chapter 4] gives a (lightweight) overview of most of the algorithms described in this chapter. Details can be found in the lecture notes [Chakrabati, 2011, Lectures 2, 4, 5, and 6], and also in [Blum et al., 2020, Chapter 7]. For the section on hashing, background can be found in many textbooks on data structures.

[Mitzenmacher and Upfal, 2005, Chapter 13] may serve as an introduction to strongly k -universal families of hash functions.

[Bar-Yossef et al., 2002] discusses various algorithms for counting distinct elements. The algorithms for computing frequency moments are from [Alon et al., 1999]. The entry [Cormode, 2009] of the Encyclopedia of Database Systems is an introduction to the Count-Min sketch and its numerous applications.