tightcenter

panikzettel.htwr-aachen.de

Algorithmic Foundations of Data Science Panikzettel™

Jan Fritz, Christoph von Oy, Sophie Hallstedt

Version 5 — 05.08.2020

Contents

0 Introduction

This is the Panikzettel for Algorithmic Foundations of Data Science. It's 'kinda' long. We also don't really know how it happened. We were going through the slides and added almost everything that could be important.

1 Machine Learning Basics

Some terminology:

- *Data* is a collection of data items.
- Each *data item* is represented by a *feature vector* of properties.
- *Properties* are also called features or attributes. Each feature has a domain of possible values.
- The *instance space* is the Cartesian product of the domains. This is exactly the space of all possible feature vectors.
- The *dimension* of the instance space is the number of features.
- The given data is often split into a training sequence and a test set. The *validation* is the evaluation of the trained model against the test data.

1.1 Types of Learning

1.1.1 Supervised Learning

The agent tries to learn functions from exemplary input-output pairs.
This is called classification if the function is finite-valued. In this case, it is the prediction of values for future inputs.
It is called regression if the function is numerical. In this case, the agent tries to predict expected values for future inputs.

In a *passive learning* scenario the training examples are given without any manual influence in the selection of examples.
In an *active learning* scenario the learning algorithm can actively choose specific data points and ask for their target values.

One distinction:

- In *batch learning* all examples are given at once and the agent has to come up with a good hypothesis.
- In *online learning* the examples are given over time which means the agent has to improve the hypothesis over time.

1.1.2 Semi-Supervised Learning

Semi-supervised learning is set-up like supervised learning but there are only a few and possible faulty examples.

1.1.3 Unsupervised Learning

The goal of this method is to detect patterns in data while no explicit feedback is supplied. The most important task in unsupervised learning is clustering.

1.1.4 Reinforcement Learning

The agent in reinforcement learning tries to find actions which maximize the reward or minimize the punishment. This is often a trial-and-error-process.

1.2 Hypotheses and Hypothesis Space

The goal in a supervised learning setting is to learn an unknown target function. A learning algorithm chooses a hypothesis h from a predefined hypothesis space $\mathcal{H}$. (For example all linear functions or all functions that can be described by a decision tree.)
The goal of a learning algorithm is to produce a hypothesis that generalizes well and approximates the target function well on all data points and not only on the training set.

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

Definition: Occam's Razor

Choose the simplest hypothesis consistent with the data

1.3 Nearest Neighbor Learning

The idea here is to predict the value of a function at a point x by looking at the known value of its neighbors. To avoid coincidences, it makes sense to look at several points close to x and then take the majority or average values of these points.
The underlying assumption is that items are close together if they have similar function values.

Definition: Metric

A metric is a function from an instance space $\mathbb{X}$ to $\mathbb{R}$ such that for all $x, y, z \in \mathbb{X}$ the following three properties apply:

- **Nonnegativity:**
  $d(x,y) \geq 0$ and $d(x,y) = 0 \iff x = y$
- **Symmetry:**
  $$d(x,y) = d(y,x)$$
- **Triangle Inequality:**
  $$d(x,z) \leq d(x,y) + d(y,z)$$

For example the Euclidean distance

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

is a common metric.

Definition: Metric Space

Let $\mathbb{X}$ be the instance space and d a metric. Then $(\mathbb{X}, d)$ is a metric space.

1.3.1 Description of Classification Problems

The goal of a classifier is to learn an unknown function f that associates a class $f(x) \in \mathbb{Y}$ with every data item $x \in \mathbb{X}$. Therefore, given labeled examples $(x_1, y_1), \ldots, (x_m, y_m)$ where each $x_i \in \mathbb{X}$ is a data item and $y_i = f(x)$ is the respective class, the learning algorithm is supposed to produce a classifier that predicts the value $f(x)$ for a new data item x.

1.3.2 The k-Nearest Neighbor Classifier

This classifier finds for each $x \in \mathbb{X}$ the k nearest neighbors of x. Then, it checks which class appears most among the neighbors of x.

Algorithm: k-Nearest Neighbor

Input: $k \in \mathbb{N}$ and $x \in \mathbb{X}$ where $\mathbb{X} = (\mathbb{X}, d)$ is the instance space.

Output: The class that appears most among the neighbors of x.

1. Find the k nearest neighbors (using d) of the point x.
2. Count which class appears most among the neighbors of x and return this class.

If there appears a tie between the numbers of classes among the neighbors of x it can be broken arbitrarily. For example, the class with the lowest index could be chosen.

The remaining question is how to choose the number of neighbors k that is taken into account. There is no definite answer to this question but there are a few rules of thumb that can be applied:

- If $k = 1$, then the hypothesis (=classifier) is guaranteed to be consistent with the examples but it is very likely to overfit.
- The larger k is chosen, the simpler the hypothesis gets but at some point the hypothesis will start to over-simplify. For example, if we choose k as high as the number of available points x_i.
- The best value of k depends on the application or can be learned by Machine Learning techniques too.

1.4 Decision Trees

Decision trees are only defined for functions that have finite-valued features and finitely many output values. If there are numerical values, they are partitioned into finitely many intervals.

1.4.1 Syntax of Decision Trees

A decision tree is a tree with labeled nodes and edges. It has the following properties:

- Every internal node is labeled with a feature.
- Every edge is labeled by a value or range of values from the feature in the source node.
- Every leaf is labeled with an output label.

The following algorithm is used to build decision trees:

Algorithm: Greedy Decision Tree Building

Input: The set of features $\mathcal{A}$ and the set of examples S.
Output: A decision tree t

if $S = \emptyset$ **then**
    create leaf t with an arbitrary value
else if all examples in S have the same result **then**
    create leaf t with that output
else
    Choose feature $A \in \mathcal{A}$ that discriminates best between examples in S ▷ How to do this is in the next chapter.
    Create new node t with feature A
    Partition examples in S according to their A-value into parts $S_1, ..., S_m$
    Recursively call algorithm in $\mathcal{A} \setminus \{A\}$ on the partitions $S_1, ..., S_m$ and attach the resulting trees t_i as children to t
end if
return t

Theorem: Complexity of Computing Decision Trees

Computing a minimal decision tree for a given set of examples is NP-complete. More precisely, the following decision problem is NP-complete:
Instance: Examples $(x_1, y_1), ..., (x_m, y_m)$ 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

1.4.2 Representation of Boolean Formulas as Decision Trees

Theorem: Boolean Formulas as Decision Trees

Let $\mathbb{B} := \{0, 1\}$ be the the Boolean domain and $f : \mathbb{B}^n \to \mathbb{B}$ be a Boolean function that can be represented by a decision tree of height k. Then f can be represented by both a k-CNF[a] and a k-DNF[b].

[a] CNF = conjunctive normal form with clauses (= disjunctions of literals) of at most k literals.
[b] DNF = disjunctive normal form with terms (=conjunctions of literals) of at most k literals.

1.5 The Perceptron

The perceptron algorithm is a linear classification algorithm. The goal is to learn an unknown target function $f : \mathbb{R}^\ell \to \{1, -1\}$. The input of the learning algorithm is a sequence of the form

$$S = ((x_1, y_1), \dots (x_m, y_m)) \in \mathbb{R}^\ell \times \{1, -1\}.$$

The hypothesis space consists of linear separators, meaning, functions $h : \mathbb{R}^\ell \to \mathbb{R}$ of the form

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

for some weight vector $w \in \mathbb{R}^\ell$ and a bias $b \in \mathbb{R}$. (Note that $\langle a, b \rangle$ denotes the scalar product.)

Formally, a linear classification problem is not realizable because the target function has the range $\{+1, -1\}$ and all hypotheses have the range $\{+1, 0, -1\}$. This can be solved by using a target function of the following form

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

Then, for every finite set of points $x_1, ..., x_n \in \mathbb{R}^\ell$ we can find a hypothesis h of the form $h(x) = \text{sgn}(\langle w', x \rangle - b')$ such that $f(x_i) = h(x_i)$.

Definition: Consistent Hypothesis

A hypothesis h is consistent with the training sequence $S = ((x_1, y_1), \ldots (x_m, y_m))$ if $h(x_i) = y_i$ for all $i \in [1, m]$.

1.5.1 Normalizing the Data Points

Definition: Homogeneous linear separator

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

Therefore a homogeneous linear separator is a linear separator without bias.

Suppose we have a training sequence $S = ((x_1, y_1), \ldots (x_m, y_m))$ with $x_i = (x_{i1}, \ldots, x_{i\ell}) \in \mathbb{R}^\ell$ and $y_i \in \{-1, 1\}$. The training sequence can be normalized by applying the transformation

$$x_i \mapsto \hat{x}_i := \frac{x_i'}{\max_{1 \leq j \leq m} \| x_j' \|}$$

to the data points x_i where $x_i' := (x_{i1}, \ldots, x_{i\ell}, 1)$. The 1 at the end is added to remove the need for the bias b. Instead, the bias can now be encoded as an additional entry in the weight vector w.

The normalized training sequence

$$\hat{S} = ((\hat{x_1}, y_1), \ldots, (\hat{x_m}, y_m))$$

has the following properties that are often very useful to work with:

- $\hat{S}$ has a homogeneous linear separator if and only if S has a linear separator.
- $0 < \| \hat{x_i} \| \leq 1$ for all $i \in [1, m]$

1.5.2 Algorithm for the Perceptron

Algorithm: Perceptron Algorithm

Input: Normalized training sequence S.
Output: Weight vector w such that the hypothesis $x \mapsto \text{sgn}(\langle w, x \rangle)$ is consistent with S.

$w \leftarrow 0$
repeat
  for all $(x, y) \in S$ **do**
    if $\text{sgn}(\langle w, x \rangle) \neq y$ **then**
      $w \leftarrow w + yx$
    end if
  end for
until $\text{sgn}(\langle w, x \rangle) = y$ for all $(x, y) \in S$

The perceptron algorithm always finds linear separators if they exist and does this very efficiently. However, the separators found are only consistent, but not optimal in any sense.

Theorem: Runtime of the perceptron algorithm

Let S be a normalized sequence of examples such that there is a homogeneous linear separator consistent with S of margin γ.
Then the perceptron algorithm applied to S finds a linear separator after at most $\frac{1}{\gamma^2}$ updates of w.

Definition: Margin of a linear separator

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

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

1.6 k-Means Clustering

The goal of k-means clustering is to put a collection of data points into k clusters. In the context of the lecture, k is fixed in advance. Clustering is an unsupervised learning problem.

Definition: Centroid Clustering Problem

Given data points $x_1, ..., x_n \in \mathbb{R}^\ell$, $k \in \mathbb{N}$, find points $z^1, ..., z^k \in \mathbb{R}^\ell$ and a partition $C^1, ..., C^k$ of $\{x_1, ..., x_n\}$ that minimizes

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

Algorithm: k-means algorithm

Input: $x_1, ..., x_n \in \mathbb{R}^\ell$, $k \in \mathbb{N}$
Output: A partition $C^1, ..., C^k$ of the data points.

Choose initial centroids $z^1, ..., z^k$
repeat
  $C^j \leftarrow \emptyset$ for all $j \in [1, k]$
  for $i \leftarrow 1$ to n **do**
    $j \leftarrow argmin_j \| x_i - z^j \|$
    add x_i to C^j
  end for
  $z^j \leftarrow \frac{\sum_{x \in C^j} x}{|C^j|}$ for all $j \in [1, k]$
until $C^1, ..., C^k$ no longer change

Theorem: Complexity of Centroid Clustering

The following properties apply for the complexity of centroid clustering:

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

Theorem: Runtime of the k-means algorithm

The k-Means algorithm always halts in a finite number of steps. This number of steps can be exponential in the number n of input points but in practice, the algorithm usually converges quickly.

The k-Means algorithm does not necessarily compute an optimal solution for the Centroid Clustering problem. The found clustering can depend on the chosen initial centroids.

2 Information and Compression

2.1 Background from Probability Theory

The lecture material begins with a refresher on random variables. For basic probability theory refer to the panikzettel on stochastics.

Theorem: Markov's Inequality

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

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

Theorem: Chebyshev's Inequality

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

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

2.1.1 Concentration Inequalities

Concentration inequalities are used to bound the probability of unlikely events (i.e. events that lie in the outer tails of a distribution). In this lecture, they are mostly used to estimate error bounds for various algorithms.

Definition: Concentration Inequalities

Let $X = \sum_{i=1}^{n} X_i$ a sum of random variables with expected value $\mu := E(X)$. Then a **concentration inequality** (also called **tail bound**) has the form

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

Theorem: Chernoff Bounds

Let $X_1, ..., X_n$ 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

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

Consequently:

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

Theorem: Hoeffding Bounds

Let $X_1, ..., X_n$ be a sequence of i.i.d. $\{0,1\}$-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

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

Consequently:

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

Theorem: Log Sum Inequality

For all $i \in [n]$, let $p_i \in \mathbb{R}_{\geq 0}$, $q_i \in \mathbb{R}_{>0}$, and let $p := \sum_i p_i$ and $q := \sum_i q_i$. Then

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

Theorem: Concentration for More General Random Variables

Let $X_1, ..., X_n$ be a sequence of independent random variables with $E(X_i) = 0$ and $Var(X_i) \leq \sigma^2$ 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 $3 \leq k \leq \lceil \frac{a^2}{4n\sigma^2} \rceil$. Then

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

2.2 Entropy

2.2.1 Information of an Event

The idea is to find a measure for the information content of a single event in a probability space so that it only depends on the probability of the event:

- An event that is certain (has probability 1) has information content 0.
- An event that is impossible (has probability 0) has no information content.
- Rarer events have higher information content.
- The joint information content of two independent events is the sum of their individual information contents.

We assign information content $I(A)$ to events A of a finite probability space $(\Omega, \mathcal{P})$ such that all our requirements are satisfied by letting $I(A) = \log_b \frac{1}{\mathcal{P}(A)}$ for some basis $b > 1$.

Definition: Information Content

The *information content* of an event A is

$$I(A) = \log_2 \left(\frac{1}{\mathcal{P}(A)} \right).$$

2.2.2 Entropy

The entropy is the expected information value of an event ω.

Definition: Entropy of a Probability Distribution

The entropy of a probability distribution $\mathcal{P}$ on a finite sample space Ω is defined as

$$H(\mathcal{P}) := \sum_{\omega \in \Omega} \mathcal{P}(\{\omega\}) \cdot \log_2 \frac{1}{\mathcal{P}(\{\omega\})}.$$

To avoid the case where the denominator is 0 we define $0 \cdot \log(\frac{1}{0}) = 0$. Alternatively sum only over the events with $\mathcal{P}(\omega) > 0$.

Definition: Entropy of a Random Variable

The entropy of a random variable X with finite range is defined as

$$H(X) := \sum_{x \in range(X)} Pr(X = x) \cdot \log_2 \frac{1}{Pr(X = x)}$$

Intuitively, it is also possible to view entropy as a measure of disarray. I.e. low entropy means that after drawing a high number of samples from a distribution we will not see much variation.

2.2.3 Entropy for Decision Tree Learning

We need a measurement for the information content of a feature in order to choose the next node in decision tree learning. Thus, we use the concept of entropy to find the feature that discriminates best.
In the decision tree setting, we have a set S of labeled examples (x, y), where x is the feature vector over $\mathcal{A}$ and $y \in \mathbb{Y}$ is the target value. With this, we can describe the information gain of a feature A as the difference between the entropies of $\mathcal{P}$ and $\mathcal{P}_{A=x}$ weighted by the relative size of $S_{A=x}$.

Definition: Information Gain

Let $H(\mathcal{P})$ be the entropy of the probability distribution and $\mathbb{D}_A$ is the set of all possibles values of the feature A. The *information gain* of feature A is then

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

Step by step:

1. Compute the total entropy $H(\mathcal{H})$.
2. Compute all $H(\mathcal{P}_{A=x})$.
3. Compute information gain and choose feature with the highest gain.

2.3 Compression

Entropy can be interpreted in two ways:

1. Information is the average number of bits that are needed to store samples from a distribution. We assume we use the best possible encoding scheme to store the information.
2. The information content of the distribution should measure how well we can compress a string consisting of independently sampled symbols.

Definition: Compression Scheme

A compression scheme over Σ is a pair $\Gamma = (com_\Gamma, dec_\Gamma)$ where $com_\Gamma : \Sigma^* \to \{0,1\}^*$ is a compression mapping and $dec_\Gamma : \{0,1\}^* \to \Sigma^*$ is a decompression mapping.
A lossless compression means that $dec(com(x)) = x$ for all $x \in \Sigma^*$.

- Intuitively, the compression rate of a scheme Γ is the maximum compression rate of Γ on all strings in Σ^*, but this maximum does not necessarily exist
- Encoding the symbols of Σ as bit strings requires $\lceil \log|\Sigma| \rceil$ bits per symbol
- One may argue that a compression scheme Γ actually achieves compression when

$$com(x) < |x| \cdot \lceil \log |\Sigma| \rceil \ \forall x$$

or equivalently

$$\rho_\Gamma(n) < \lceil \log |\Sigma| \rceil.$$

Definition: Compression Rate

Compression rate of scheme Γ on string x:

$$\frac{|com_\Gamma(x)|}{|x|}.$$

As a function, the compression rate of Γ is $\rho_\Gamma : \mathbb{N} \to \mathbb{R}$ defined as

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

Theorem: Existence of Lossless Compression

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

2.3.1 Generating the Input Strings

Assume that input strings are generated randomly and symbols x_i in a string $x = x_1 ... x_n$ are independently identically distributed (i.i.d.), i.e. each symbol of a string is drawn individually from the same distribution.
This does not give us a probability distribution on the set Σ^* of all strings over Σ.

Definition: Input Strings for Compression Schemes

For every $n \in \mathbb{N}$ define a probability distribution $\mathbb{P}^n$ on Σ^n as

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

Strings of length n are distributed according to $\mathcal{P}^n$.

2.4 Lossy Compression

Definition: Loss Rate

Let Γ be a compression scheme over Σ. The loss rate of Γ is the probability that a compressed string is not decompressed correctly:

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

Key idea: Only focus on strings that occur significantly often, while ignoring unlikely strings.

It is possible to define a compression scheme $\Gamma_\epsilon = (com_\epsilon, dec_\epsilon)$ with an upper limit $\epsilon > 0$ for the loss rate during compression:

1. For every $n \in \mathbb{N}$, choose a set $S_\epsilon(n) \subseteq \Sigma^n$ of minimum cardinality such that
$$\mathcal{P}^n(S_\epsilon(n)) \geq 1 - \epsilon$$
Let $s_\epsilon(n) := \lceil \log(|S_\epsilon(n)|) \rceil$.
2. Define the compression mapping com_ϵ so that for every n we have
$$com_\epsilon(\Sigma^n) \subseteq \{0,1\}^{s_\epsilon(n)}$$
and the restriction of com_ϵ to $S_\epsilon(n)$ is injective.
3. Define the decompression mapping dec_ϵ so that for all $x \in S_\epsilon(n)$ we have $dec_\epsilon(com_\epsilon(x)) = x$

Resulting **loss rate**

$$\lambda_\epsilon(n) \leq \epsilon \; \forall n \in \mathbb{N}$$

and **compression rate**

$$\rho_\epsilon(n) = \frac{s_\epsilon(n)}{n} \; \forall n \in \mathbb{N} \quad \text{with} \quad \lim_{n\to\infty} \frac{s_\epsilon(n)}{n} = H(\mathcal{P})$$

2.5 Shannon's Source Coding Theorem

Theorem: Shannon's Source Coding Theorem

1. For every $\epsilon > 0$ there is a compression scheme Γ_ϵ over Σ such that $\lambda_{\Gamma_\epsilon,\mathcal{P}}(n) \leq \epsilon$ for all n and $\lim_{n\to\infty} \rho_{\Gamma_\epsilon}(n) = H(\mathcal{P})$.
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 $n \in \mathbb{N}$.

3 Statistical Learning Theory

3.1 PAC (Probably Approximately Correct) Learning Framework

Goal: Given training examples, we want to "learn" a hypothesis that generalises well (i.e. is a good approximation to the unknown target function).

Definition: Formal framework for PAC

- *Instance space* $\mathbb{X}$,
- *Data generating probability distribution* $\mathcal{D}$ on $\mathbb{X}$,
- *Target function* $f^\star : \mathbb{X} \to \{0,1\}$,
- *Training sequence* $T = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathbb{X} \times \{0,1\})^m$,
- *Hypothesis* $h : \mathbb{X} \to \{0,1\}$.

Definition: Training error

The training error (or hypothetical risk) of a hypothesis h w.r.t a training sequence T is

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

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

If the instances $x_1, \ldots, x_m$ of a training sequence are drawn independently from $\mathcal{D}$ we write $T \sim \mathcal{D}^m$.

Definition: Generalization error

Let f^* be the target function. The generalization error of a hypothesis h is

$$\mathrm{err}_{\mathcal{D}}(h) = \Pr_{x \sim \mathcal{D}}(h(x) \neq f^\star(x))$$

Definition: 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 $\mathbb{X}$

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

In practice, the training error is easier to compute, leading to ERM algorithms.

Definition: Empirical Risk Minimization

An algorithm that returns on input T a hypothesis h_T in a given hypothesis class $\mathcal{H}$ is an ERM algorithm if

$$h_T = \operatorname*{argmin}_{h \in \mathcal{H}} \mathrm{err}_T(h)$$

Definition: Regularization

To avoid overfitting, we expand the definition of ERM to the following formula, using an arbitrary monotone (often linear) function $\rho(h)$:

$$h_T = \operatorname*{argmin}_{h \in \mathcal{H}} (\mathrm{err}_T(h) + \rho(\mathrm{cost}(h)))$$

3.2 Sample Size Bounds for Finite Hypothesis Classes

Since a learning algorithm can only see the training error, we need to aim for situations in which the training error is close to the generalization error. We can prove that for sufficiently large sample sizes, a low training error leads to a small generalization error.

Definition: Agnostic Learning

A learning approach that does **not** assume that the learning problem at hand is realisable.

Note $\ln(x) = \log_e(x)$.

Theorem: Simple Sample Size Bound

Let $\mathcal{H}$ be finite, $\varepsilon, \delta > 0$ and

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

Then for any data generation distribution $\mathcal{D}$

$$\Pr_{T \sim \mathcal{D}^m} (\forall h \in \mathcal{H} : (\text{err}_T(h) = 0 \Rightarrow \text{err}_{\mathcal{D}}(h) \leq \varepsilon)) > 1 - \delta$$

If we take an ERM algorithm with a finite hypothesis space that satisfies the realizability assumption and define m according to the simple sample size bound, then we end up with a PAC algorithm.

If we cannot make any assumptions about realizability, the following two bounds come into play.

Theorem: Uniform Convergence

Let $\mathcal{H}$ be finite, $\varepsilon, \delta > 0$ and

$$m \geq \frac{1}{2\varepsilon^2} \log(\frac{2|\mathcal{H}|}{\delta})$$

Then for any data generation distribution $\mathcal{D}$

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

Theorem: Agnostic PAC Learning Sample Size Bound

Consider an ERM algorithm with a finite hypothesis class $\mathcal{H}$. Let $\varepsilon, \delta > 0$ and

$$m \geq \frac{2}{\varepsilon^2} \log(\frac{2|\mathcal{H}|}{\delta})$$

Then for any data generation distribution $\mathcal{D}$ and $h^\star = \text{argmin}_{h \in \mathcal{H}} \text{err}_D(h)$

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

3.3 Infinite Hypothesis Classes

3.3.1 Description Schemes

Definition: Description Scheme

- Let $\mathcal{H}$ be a hypothesis class, it can be infinite.
- Let Δ be a scheme to describe hypotheses with strings built from a finite alphabet Σ.
- For every $h \in \mathcal{H}$ let $|h|_\Delta$ be the length of the shortest description.

Theorem: Sample Size Bounds for Infinite Hypothesis Classes

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 $\mathcal{D}$,

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

Note that the theorem does not depend on the description scheme. Note further that the theorem only says that simple hypotheses are never bad, not that more complex hypotheses are worse than simpler ones.

3.3.2 VC Dimension

The second generalization to infinite hypothesis classes is a combinatorial measure for the complexity of a hypothesis class.

Definition: VC Dimension

- Let $\mathcal{H}$ be a hypothesis class of functions $h : \mathbb{X} \to \{0, 1\}$
- A subset $Y \subseteq \mathbb{X}$ is **shattered** by $\mathcal{H}$ if every function $g : Y \to \{0, 1\}$ is the restriction of a function in $\mathcal{H}$ to Y.
- The **VC-dimension** $\mathrm{VC}(\mathcal{H})$ is the size of the largest set shattered by $\mathcal{H}$ or ∞ if arbitrarily large sets are shattered.

Theorem: Uniform Convergence for VC Dimension

Let $\mathcal{H}$ be a hypothesis class of finite VC-dimension d. Let $\varepsilon, \delta > 0$ and

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

for a suitable constant c. Then for any data generation distribution $\mathcal{D}$,

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

Most exercises for VC dimension proofs consist of the same three parts.

1. Claim $VC(\mathcal{H}) = d$. There is no formal concept to determine a correct d. It can be a bit of guessing and checking.
2. Show $VC(\mathcal{H}) \geq d$ by showing there is a set of size d that can be shattered by $\mathcal{H}$. Construct a set with d elements and for each configuration of 1 and 0 of this set show that there exists a corresponding function in $\mathcal{H}$.
3. Show $VC(\mathcal{H}) \leq d$ by showing that no set of size $d + 1$ can be shattered by $\mathcal{H}$.

Another explanation with examples can also be found here.

4 Multiplicative Weight Updates

This section contains different Multiplicative Weight Update (MWU) Algorithms. First for boolean events and then a generalized version for multiple events.

4.1 MWU Algorithms

4.1.1 Deterministic MWU Algorithm

For this algorithm, we assume a setting with binary events representing the price movements of a stock (up or down). We have a set of n experts that give out binary advice. The goal of the algorithm is to minimize our loss by weighing the expert's advice and following the weighted majority.

Definition: Weight Majority Notation

We have n experts numbered $1, \dots, n$ and define for every $t \geq 1$:

- $p^{(t)} \in \{0,1\}$: Price movement on day t (0 for down, 1 for up),
- $a_i^{(t)} \in \{0,1\}\, \forall i \in [n]$: Advice of expert i on day t (0 for don't by, 1 buy),
- $l_i^{(t)} = \sum_{s=1}^{t} |a_i^{(s)} - p^{(s)}| \forall i \in [n]$: Cumulated loss of expert i after t days,
- $d^{(t)} \in \{0,1\}$: Our decision on day t (0 for don't buy, 1 for buy),
- $l^{(t)} = \sum_{s=1}^{t} |d^{(s)} - p^{(s)}|$: Our cumulated loss after t days,
- $w_i^{(t)} \forall i \in [n]$: The weight assigned by the algorithm to every expert.

Algorithm: Weighted Majority Algorithm

For some constant $0 < \alpha \leq 0.5$, we initially assign weights: $w_i^{(1)} = 1 \forall i \in [n]$. We then, for every $t \geq 1$, compute our decision based on the previous weights and update the weights based on the loss:

$$d^{(t)} = \begin{cases} 1 & \text{if} \sum_{\substack{i \in [n] \\ a_i^{(t)}=1}} w_i^{(t)} \geq \sum_{\substack{i \in [n] \\ a_i^{(t)}=0}} w_i^{(t)} \\ 0 & \text{otherwise} \end{cases} \qquad w_i^{(t+1)} = \begin{cases} w_i^{(t)} & \text{if } a_i^{(t)} = p^{(t)} \\ (1-\alpha) w_i^{(t)} & \text{otherwise} \end{cases}$$

Theorem: Weighted Majority Analysis

For every $t \geq 1$ and every $i \in [n]$,

$$l^{(t)} \leq \frac{2 \ln n}{\alpha} + 2(1+\alpha) l_i^{(t)}.$$

This theorem guarantees that our cumulated loss is bounded from above by twice the cumulated loss of the best expert.

4.1.2 Randomized MWU Algorithm

This is a generalization to multiple events. Instead of simply following the majority vote of experts, we draw an expert randomly from a probability distribution.

Definition: Multiplicative Weight Update Notation

- I: Set of n experts, usually $I = [n]$,
- J: Set of possible events,
- $L \in \mathbb{R}^{I \times J}$: Loss matrix where L_{ij} describes the loss of following expert i when event j happens, usually normalized.

We define for every $t \geq 1$:

- $j^{(t)} \in J$: Events that happen at time t,
- $w_i^{(t)} \forall i \in I$ The weight assigned by the algorithm to expert i,
- A probability distribution $\mathcal{D}^{(i)}$ on I defined as

$$\mathcal{D}^{(t)}(\{i\}) = p_i^{(t)} = \frac{w_i^{(t)}}{\sum_{i' \in I} w_{i'}^{(t)}},$$

- $L^{(t)} = \sum_{i \in I} p_i^{(t)} L_{ij^{(t)}}$: Our expected loss when choosing the expert according to the probability distribution.

Algorithm: Randomized MWU Algorithm

For some constant $0 < \alpha < 1$, we initially assign weights: $w_i^{(1)} = 1$ for all $i \in I$. We then, for every $t \geq 1$, update the weights based on the loss:

$$w_i^{(t+1)} = (1-\alpha)^{L_{ij^{(t)}}} w_i^{(t)}$$

Theorem: Multiplicative Weight Update Algorithm Analysis

For ever $t \geq 1$ and every $i \in I$,

$$\sum_{s=1}^{t} L^{(s)} \leq \frac{\ln n}{\alpha} + (1+\alpha) \sum_{s=1}^{t} L_{ij^{(s)}}.$$

This theorem guarantees that we have an upper bound on the expected loss over all time-steps independent of the happening events.

4.2 Boosting Weak Learning Algorithms

This section will present some techniques to improve the performance of classification algorithms using Multiplicative Weight Updates.

Definition: Strong Learner

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

$$\Pr_{T \sim \mathcal{D}^m}(err_{\mathcal{D}}(h_T) \leq \epsilon) > 1 - \delta.$$

Definition: Weak Learner

Let $0 \leq \gamma < \frac{1}{2}$. A learning algorithm that produces a hypothesis h_T on an input T is a weak learning algorithm with error parameter γ if for all $\delta > 0$ there is an $m = m(\delta)$ such that for every probability distribution $\mathcal{D}$ on $\mathbb{X}$

$$\Pr_{T \sim \mathcal{D}^m}(err_{\mathcal{D}}(h_T) \leq \gamma) > 1 - \delta.$$

The idea of boosting is reducing the error of a weak learner to turn it into a strong learner. To do

this the following steps are executed:

1. Draw a random subset from the initial training set. Each subset is drawn from a different probability distribution.
2. Run the weak learner on this subset.
3. Adapt the distribution using multiplicative weight updates.

This concept is called AdaBoost.

Definition: Boosting Problem

Input: A sufficiently long training sequence $T = ((x_1, y_1), ..., (x_n, y_n)$ and an error parameter $\epsilon > 0$.
Output: A Hypothesis h with $err_T(h) < \epsilon$.

Theorem: Consistent Hypotheses for Boosting

Let n be the length of the training sequence and ϵ be the error parameter in the boosting problem. If $\epsilon < \frac{1}{n}$ then the resulting hypothesis h is consistent with the training sequence.

The boosting algorithm consists of three parts. The setup for the weak learner, the setup for the MWU algorithm and the actual boosting.

4.2.1 Setup for the Weak Learner

Let $\mathcal{L}$ be the weak learner and γ its error parameter.

- Run $\mathcal{L}$ with the the confidence parameter $\delta_0 = \frac{1}{10}$ and let $m_0 = m(\delta_0) \leq n$ for $n \in \mathbb{N}$ be the number of examples that are needed.
- Identify the 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 $E \subseteq \mathbb{X}$. From now on $\mathcal{D}$ and $\mathcal{D}_X$ are the same.
- The weak learner $\mathcal{L}$ will now only run on examples drawn accordingly to the already known probability distributions $\mathcal{D}$ on X.
- We call a concluded hypothesis good if it has a generalization error smaller than γ. The weak learner generates such a good hypothesis with a probability of at least $1 - \delta_0 = 0.9$.
- In case $\mathcal{L}$ generated a bad hypothesis we re-run it on new examples until a good hypothesis occurs. With an extremely high probability, this requires only a small number of re-runs.

4.2.2 Setup for the MWU Algorithm

Let the set of experts be $I = [1, n]$ and the set J of the events be the set of hypotheses generated by the weak learner $\mathcal{L}$ when presented with m_0 input examples from X.
Then the loss matrix is defined as:

$$L_{i,j} = \begin{cases} 1 & \text{if } j(x_i) = y_i \\ 0 & \text{else} \end{cases}$$

The loss for the expert i is positive and the weight will be decreased if a hypothesis is correct for x_i. Finally, we use the update parameter $\alpha = \frac{1}{2} - \gamma$.

4.2.3 The Boosting Algorithm

We use the described setups for the weak learner $\mathcal{L}$ and the MWU algorithm. As before, ϵ is the error parameter in the input of the boosting algorithm and α is the update parameter from the setup of the MWU algorithm.

Algorithm: Boosting Algorithm

1. Consider a run of the MWU algorithm where $j^{(s)}$ is a hypothesis obtained by running $\mathcal{L}$ on $\mathcal{D}^{(s)}$ until it returns a good hypothesis.
2. Run the MWU algorithm for $t = \frac{2}{\alpha^2} \cdot \ln(\frac{1}{\epsilon})$ rounds.
3. The final hypothesis h is defined by

$$h(x) = \begin{cases} 1 & \text{if } |\{s \leq t \mid j^{(s)}(x) = 1\}| \geq \frac{t}{2} \\ 0 & \text{else} \end{cases}.$$

4. Return h.

Theorem: Error of the Returned Hypothesis from the Boosting Algorithm

Let ϵ be the error parameter in the input of the boosting algorithm. The error of the hypothesis that is returned from the boosting algorithm is

$$err_T(h) < \epsilon.$$

4.2.4 Run-time of the Boosting Algorithm

The running time of the boosting algorithm largely depends on the calls to the weak learner $\mathcal{L}$. If $\mathcal{L}$ has a short running time, then the boosting algorithm has a short running time too.

For the number of rounds: If we want that the final hypothesis classifies all examples correctly, we need $\epsilon \approx \frac{1}{n}$ steps. Therefore the MWU algorithm is executed in $\mathcal{O}(\log n)$ rounds.

4.3 Bandit Learning

The bandit learning problem is a reinforcement learning problem. In it we have a set of n slot machines also called one-armed bandits. All the machines have a different internal setting. Therefore some of them give a higher reward (= more money) on average than others.
In each round, we choose one of the machines with a certain strategy and observe the reward. Such a strategy could be simply a randomized strategy. The goal is to minimize the difference (called regret) between the total payoffs of our strategy and the payoff of the best machine.
We also assume that the setting is *adversarial*. Thus, an adversary fixes the payoff for each machine in each round in a way that maximizes our regret.

4.3.1 Formal Description of Bandit Learning

In this setting we have n slot machines numbered from 1 to n. These slot machines represent the actions. For every $s \geq 1$ and every $a \in [1, n]$, there is a reward $q_a^{(s)}$ with $0 \leq q_a^{(s)} \leq 1$. With this we can describe a payoff matrix $Q := (q_a^{(s)})_{a \in [1,n], s \geq 1}$. If the number of rounds t is fixed in advance Q is a $n \times t$ matrix.

Definition: Reward and Regret of a Sequence

Let $a = (a^{(1)}, ..., a^{(t)}) \in [1,n]^t$ be a sequence of actions. The reward of a is

$$q(a) := \sum_{s=1}^{t} q_{a^{(s)}}^{(s)}.$$

The regret of a is

$$r(a) := q_{\max}^{(t)} - q(a).$$

The choice in such a strategy can be random. Then, $a^{(t)}$ is drawn according to some probability distribution $\mathcal{D}^{(t)}$ on $[1,n]$.

Definition: Maximal Single-Action Reward

The maximal single-action reward after round t is

$$q_{\max}^{(t)} := \max_{a \in [1,n]} \sum_{s=1}^{t} q_a^{(s)}.$$

Definition: Strategy

A strategy or algorithm A picks an action $a^{(t)}$ on each round t only depending on the actions $a^{(s)}$ and the rewards $q^{(s)} := q_{a^{(s)}}^{(s)}$ of the previous rounds $s = 1, ..., t-1$.

Definition: Reward and Regret of a Strategy

The expected reward of a strategy A at time t is

$$q(A) := E(q(a^{(1)}, ..., a^{(t)})).$$

The regret of A is

$$r(A) := E(r(a^{(1)}, ..., a^{(t)})).$$

4.3.2 Multiplicative Weights Update Algorithm

Exp3 stands for exponential-weight algorithm for exploration and exploitation.

Algorithm: Exp3

Parameter: γ with $0 < \gamma \leq 1$ to determine the tradeoff between exploration and exploitation.
Initialization: $w_a^{(1)} = 1$ for all $a \in [1,n]$.

1: **for** $s = 1, 2, ..., t$ **do**
2: $\mathcal{D}^{(s)}$ is the probability distribution defined by

$$\Pr_{\mathcal{D}^{(s)}}(\{a\}) := p_a^{(s)} := (1-\gamma)\frac{w_a^{(s)}}{\sum_{a'=1}^{n} w_{a'}^{(s)}} + \frac{\gamma}{n}$$

3: Draw action $a^{(s)}$ randomly from $\mathcal{D}^{(s)}$
4: $q^{(s)} \leftarrow q_{a^{(s)}}^{(s)}$ ▷ The reward
5: Update the weights:

$$w_a^{(s+1)} \leftarrow \begin{cases} w_a^{(s)} \cdot \exp\left(\frac{\gamma q^{(s)}}{n p_a^{(s)}}\right) & \text{if } a = a^{(s)} \\ w_a^{(s)} & \text{else} \end{cases}$$

6: **end for**

Theorem: Maximal Regret of Exp3

For every payoff matrix, the expected regret of the Exp3 algorithm is bounded by

$$r(Exp3) \leq (e-1) \cdot \gamma \cdot q_{\max}^{(t)} + \frac{1}{\gamma} \cdot n \cdot \ln(n).$$

This can be improved as follows. 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 Exp3 with the parameter γ^* satisfies

$$r(Exp3) \leq 2.63 \cdot \sqrt{q_{\max}^{(t)} \cdot n \cdot \ln(n)}.$$

5 High Dimensional Data

5.1 The Strange Geometry of High-Dimensional Spaces

Definition: Volume of High Dimensional-Objects

Let $X \subseteq \mathbb{R}^\ell$. Then $vol(X)$ is the volume of X. It is only defined for measurable sets.

Theorem: Properties of High-Dimensional Objects

- Let $X \subseteq \mathbb{R}^\ell$, let $c \in \mathbb{R}$ and let $cX := \{cx \mid x \in X\}$. Then
$$vol(cX) = c^\ell vol(X).$$
- Let $X \subseteq \mathbb{R}^\ell$ such that $vol(X) > 0$ and let $0 \leq \epsilon \leq 1$. Then
$$\frac{vol((1-\epsilon)X)}{vol(X)} \leq e^{-\epsilon\ell}.$$

5.1.1 The High-Dimensional Unit Ball

Definition: Unit Ball

The ℓ-dimensional unit ball is the set

$$B^\ell := \{x \in \mathbb{R}^l \mid\mid\mid x \mid\mid\mid \leq 1\}$$

Theorem: Volume of the Unit Ball

The volume of the ℓ-dimensional unit ball is

$$\lim_{\ell \to \infty} vol(B^\ell) = 0.$$

For a fixed ℓ the volume of the unit ball is defined approximately. The unit ball is covered by $2k$

cylinders of different size. Then we have an approximation for the volume by

$$vol(B^\ell) \leq \left(\frac{2}{k} \sum_{i=1}^{k} \cos\left(\frac{i-1}{k} \right)^{\ell-1} \right) \cdot vol(B^{\ell-1}).$$

The unit ball is an example of why high-dimensional objects can be strange. The volume of the unit ball does increase with a higher number of dimensions at first but after the jump from five to six dimensions, the volume decreases.

The theorems on the right mean that at least a $(1 - \frac{2}{c} \cdot e^{\frac{-c^2}{2}})$-fraction of a unit ball has a distance of at most $\frac{c}{\sqrt{\ell-1}}$ from the equator of the unit ball.

The high-dimensional unit ball has some properties which are similar to the properties of probability distributions. The volume of the unit ball is concentrated near the equator. A similar effect occurs when drawing a point $x = (x_1, ..., x_\ell) \in B^\ell$ from a probability distribution. On average, the values $|x_i|$ will be around $\frac{1}{\sqrt{\ell}}$ because otherwise the length of $\|x\|$ would be too large. Or in other words the probability of an $|x_i|$ being much larger than $\frac{1}{\sqrt{\ell}}$ is very low.

Theorem: Volume Concentration at the Equator

Let $\ell \geq 3$ and $c \geq 1$. Then

$$\frac{vol\left(\left\{x \in B^\ell \mid |x_1| > \frac{c}{\sqrt{\ell-1}}\right\}\right)}{vol(B^\ell)} \leq \frac{2}{c} \cdot e^{\frac{-c^2}{2}}$$

Theorem: Unit Vector and Volume

Let $\ell \geq 3$, $c \geq 1$ and $a \in \mathbb{R}^\ell$ a unit vector ($\|a\| = 1$). Then we have:

$$\frac{vol\left(\left\{x \in B^\ell \mid |\langle a, x\rangle| > \frac{c}{\sqrt{\ell-1}}\right\}\right)}{vol(B^\ell)} \leq \frac{2}{c} \cdot e^{\frac{-c^2}{2}}$$

5.2 Dimension Reduction by Random Projections

It is possible to reduce the dimension of a set of points in a high-dimensional Euclidean space while approximately preserving the distance between all pairs of points in the set. In many scenarios, lower dimension data is easier to handle computationally.

Definition: Spherical Gaussian Distribution

An ℓ-dimensional Gaussian distribution with mean $\mu \in \mathbb{R}^\ell$ and variance σ^2 in each direction is the probability distribution on $\mathbb{R}^\ell$ with density

$$p(x) = \frac{1}{(2\pi)^{\frac{\ell}{2}} \cdot \sigma^\ell} \cdot \exp\left(-\frac{\| x - \mu \|^2}{2\sigma^2} \right)$$

Note that the spherical Gaussian distribution is a special case of the multivariate normal distribution where the coordinates are independent and have the same variance.

Theorem: Construction of Spherical Gaussian Distributions

The ℓ-dimensional spherical Gaussian distribution with mean $\mu = (\mu_1, ..., \mu_\ell) \in \mathbb{R}^\ell$ and variance σ^2 in each direction can be created by drawing the coordinates x_i of $x = (x_1, ..., x_\ell)$ independently according to a normal distribution with mean μ_i and variance σ^2.

In a typical one-dimensional Gaussian distribution, most of the probability mass is near the mean. This is not the case for high-dimensional spherical Gaussians because their probability mass is concentrated in an annulus (or 'hill') of radius $\sigma^2\sqrt{\ell}$ around the mean.

Theorem: Gaussian Annulus Theorem

Let $b \leq \sqrt{\ell}$ and let $x \in \mathbb{R}^\ell$ be drawn from an ℓ-dimensional spherical Gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. Then

$$Pr(\sqrt{\ell} - b < \|x\| < \sqrt{\ell} + b) \geq 1 - 3e^{-cb^2}$$

where $c > 0$ is a constant that does not depend on ℓ and b.

The reduction mapping is used to map the high-dimensional data to a lower dimension.

Definition: Reduction Mapping

Let $k, \ell \in \mathbb{R}$ with $k \leq \ell$. The vectors $u_1, ..., u_k \in \mathbb{R}^\ell$ are drawn independently from the ℓ-dimensional spherical Gaussian distribution with mean 0 and variance 1 in each direction. Then we define the matrix U as:

$$U := \frac{1}{\sqrt{k}} \begin{pmatrix} u_1^T \\ \vdots \\ u_k^T \end{pmatrix} \in \mathbb{R}^{k \times \ell}$$

Then the mapping $x \mapsto Ux$ is the random projection that is used for the dimension reduction.

Theorem: Random Projection Theorem

Let $k, \ell \in \mathbb{R}$ with $k \leq \ell$.
For all $x \in \mathbb{R}^\ell$ and all $\epsilon > 0$ we have

$$Pr\left(|\|Ux\| - \|x\|| > \epsilon\|x\|\right) \leq 3e^{-c\epsilon^2 k}$$

where U is the matrix from the reduction mapping and c is the constant from the Gaussian Annulus Theorem.

Theorem: Corollary from the Random Projection Theorem

Let $k, \ell \in \mathbb{R}$ with $k \leq \ell$. For all $x, y \in \mathbb{R}^\ell$ and all $\epsilon \in (0, 1)$ we have

$$Pr\left((1-\epsilon)\|x-y\| \leq \|Ux - Uy\| \leq (1+\epsilon)\|x-y\|\right) \geq 1 - 3e^{-c\epsilon^2 k}.$$

Theorem: Johnson-Lindenstrauss Lemma

Let $0 < \epsilon < 1$ and $k, \ell, n \in \mathbb{N}$ such that $k \geq \frac{3}{c\epsilon^2} \cdot \ln n$ where c is the constant from the Gaussian Annulus Theorem. Then for every set $X \subseteq \mathbb{R}^\ell$ of size $|X| = n$ we have

$$Pr\left(\forall x, y \in X : (1-\epsilon)\|x-y\| \leq \|Ux - Uy\| \leq (1+\epsilon)\|x-y\|\right) \geq 1 - \frac{3}{2n}.$$

This Lemma describes the probability that the distance between two points from a set of points X is still below a certain difference after the dimension reduction.

5.3 Eigenvalues and Eigenvectors

We do not repeat all the basics for Eigenvalues, Eigenvectors and diagonalisable matrices here. They can be found in the Panikzettel for Linear Algebra.

Reminder: A matrix $A \in \mathbb{C}^{n\times n}$ is *diagonalisable* if there are is a non-singular matrix U and a diagonal matrix Λ such that $U^{-1}AU = \Lambda$.

Theorem: Relation Between Diagonalisable Matrices and Eigenvectors

Let $A \in \mathbb{C}^{n\times n}$.

1. If $U^{-1}AU = \Lambda$ where $\Lambda = diag(\lambda_1, ..., \lambda_n)$ then $\lambda_1, ..., \lambda_n$ is the spectrum (= set of eigenvalues) of A. Moreover the columns $u_1, ..., u_n$ are the eigenvectors of A associated with $\lambda_1, ..., \lambda_n$.
2. If the preconditions a to d are met then $U^{-1}AU = \Lambda$.
   a) $\lambda_1, ..., \lambda_n$ are the eigenvalues of A
   b) $u_1, ..., u_1$ is a basis of corresponding eigenvectors
   c) $U \in \mathbb{C}^{n\times n}$ is the matrix with columns $u_1, ..., u_n$
   d) $\Lambda = diag(\lambda_1, ..., \lambda_n)$

Reminder: A matrix $U \in \mathbb{R}^{n\times n}$ is *orthogonal* if $U^{-1} = U^T$ (equivalently, if the columns of U form an orthonormal basis or $\mathbb{R}^n$)

Theorem: Properties of a symmetric matrix

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.

Theorem: Spectral Decomposition

Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix. Then there is an orthogonal matrix $U \in \mathbb{R}^{n\times n}$ such that

$$A = U\Lambda U^T$$

where Λ is the diagonal matrix whose diagonal entries form the spectrum of A

5.3.1 Perron-Frobenius Theorem

Definition: Irreducible Matrix

We associate a graph G_A with each matrix $A = (A_{ij}) \in \mathbb{R}^{n\times n}$. The vertex set is $V(G_A) := [1, n]$ and the edge set is is

$$E(G_A) := \{(i, j) \mid A_{ij} \neq 0\}.$$

The matrix A is irreducible if G_A is strongly connected (= every vertex is reachable from every other vertex).

Theorem: Perron-Frobenius

Let $n \geq 2$ and let $A \in \mathbb{R}^{n \times n}$ be non-negative and irreducible with spectral radius (= maximal absolute value of an eigenvalue) $\rho = \rho(A)$. Then

1. ρ is an eigenvalue of A of algebraic multiplicity 1.
2. For all eigenvalues $\lambda \neq \rho$ of A it holds that $\rho > |\lambda|$. (Definition of spectral radius.)
3. There is unique eigenvector $u \in \mathbb{R}^n$ associated with ρ such that $||u|| = 1$ and all entries of u are positive. u is called right *Perron vector* of A.
4. There is a unique vector $v \in \mathbb{R}$ such that $v^T A = \rho v^T$ and $||v|| = 1$ and all entries of v are positive. v is called left *Perron vector* of A.

Definition: Permutation of a Matrix

Let $A \in \mathbb{R}^{n \times n}$. For a permutation π of $[1, n]$ we let A^π be the matrix with entries $A^\pi_{ij} := A_{\pi^{-1}(i)\pi^{-1}(j)}$.

Definition: Reducible Matrix

Let $A \in \mathbb{R}^{n \times n}$. A is reducible if there is a $k \in [1, n-1]$ and

- a matrix $B \in \mathbb{R}^{k \times k}$,
- a matrix $B \in \mathbb{R}^{k \times (n-k)}$,
- a matrix $C \in \mathbb{R}^{(n-k) \times (n-k)}$ and
- a permutation π of $[1, n]$ such that

$$A^\pi = \begin{pmatrix} B & C \\ 0 & D \end{pmatrix}.$$

Theorem: Reducibility of Non-Negative Matrices

Let I be the identity matrix. For every non-negative matrix $A \in \mathbb{R}^{n \times n}$ the following properties are equivalent.

- A is irreducible.
- A is not reducible.
- $(A + I)^{n-1}$ has only positive entries.

Theorem: Limit Theorem for Non-Negative Matrices

Let $n \geq 2$, let $A \in \mathbb{R}^{n \times n}$ be non-negative and irreducible with spectral radius ρ and let u, v the the Perron vectors of A. Then

$$\lim_{k \to \infty} \frac{1}{k} \sum_{i=1}^{k} \frac{A^i}{\rho^i} = \frac{1}{\langle u, v \rangle} u \cdot v^T.$$

5.4 Power Iteration

The power iteration algorithm is used to approximate an eigenvector of a matrix. It can only be used if the two following assumptions are met. Let $A \in \mathbb{C}^{n \times n}$ be a matrix with spectrum $\lambda_1, ...\lambda_n$ where $|\lambda_1| \geq |\lambda_2| \geq ... \geq |\lambda_n|$ and let $\Lambda = diag(\lambda_1, ..., \lambda_n)$. We assume:

1. $\lambda_1 \in \mathbb{R}_{\geq 0}$ and $\lambda_1 > |\lambda_2|$.
2. A is diagonisable.

The rate of convergence for the sequence $(v_k)_{k \geq 0}$ is determined by

$$\frac{|\lambda_2|}{\lambda_1} = \max_{i \geq 2} \frac{|\lambda_i|}{\lambda_1}.$$

Algorithm: Power Iteration Algorithm

Input: Matrix $A \in \mathbb{C}^{n \times n}$, vector $x \in \mathbb{C}^n$
Output: Vector $v \in \mathbb{C}^n$

1: $v_0 \leftarrow \frac{x}{\|x\|}$
2: $k \leftarrow 0$
3: **repeat**
4: $\quad k \leftarrow k + 1$
5: $\quad v_k \leftarrow \frac{A v_{k-1}}{\|A v_{k-1}\|}$
6: **until** Sequence converges (up to the required precision).
7: **return** v_k

For the algorithm to work, the initial vector x cannot be orthogonal to the target eigenvector. When choosing x randomly, however, the probability of x being orthogonal is small.

5.5 Principal Component Analysis

Let $a_1, ..., a_n \in \mathbb{R}^\ell$ be a *set of data points*. (Each entry of one of this data points is a feature.) Such sets of data points are represented by a *data matrix* $A \in \mathbb{R}^{n \times \ell}$ whose rows are $a_1^T, ..., a_n^T$.
The data matrix is called *centred* if the mean of its data points is zero. Each data matrix can be centered by replacing each data point a_i by

$$a_i' := a_i - \sum_{j=1}^{n} a_j = 0$$

5.5.1 PCA-Transformation

Definition: PCA-Transformation

Let $A \in \mathbb{R}^{n \times \ell}$ be a data matrix with rows $a_1^T, ..., a_n^T$. A PCA-transformation of A is an orthogonal matrix $U \in \mathbb{R}^{\ell \times \ell}$ with columns $u_1, ..., u_\ell$ satisfying

$$u_j = \underset{\substack{x \in \mathbb{R}^\ell \\ ||x||=1 \\ x \perp u_1,..,u_{j-1}}}{\operatorname{argmax}} \sum_{i=1}^{n} \langle a_i, x \rangle^2.$$

The lines $\mathbb{P}_i = span(u_i)$ are called the principal components of A with respect to U.

5.5.2 Best-Fit Subspaces

Definition: Best-Fit Subspace

Let $1 \leq k \leq \ell$ and $A \in \mathbb{R}^{n \times \ell}$ be the centred data matrix with rows $a_1^T, ..., a_n^T$.
A *best-fit k-dimensional subspace* for the data is a k-dimensional linear subspace $\mathbb{X} \subseteq \mathbb{R}^l$ such that the projection of $a_1, ..., a_n$ into $\mathbb{X}$ has maximum variance.

Or, a best-fit k-dimensional subspace is a subspace $\mathbb{X} = span(x_1, ..., x_k)$ where $x_1, ..., x_k$ is an orthonormal system maximizing

$$\sum_{i=1}^{n} \sum_{j=1}^{k} \langle a_i, x_j \rangle^2.$$

Theorem: PCA and Best-Fit Subspaces

Let $A \in \mathbb{R}^{n \times \ell}$ and let U be its PCA-transformation. Then for every $k \in [1, \ell]$

$$\mathbb{U}_k = span(u_1, ..., u_k)$$

is a best fit subspace for A.

This is used for dimension reduction because the projection of data into its best-fit subspace is an optimal k-dimensional approximation to the original data.

5.5.3 The Covariance Matrix and Spectral Decomposition

Definition: Covariance Matrix

Let $A \in \mathbb{R}^{n \times \ell}$ be a data matrix.
The covariance matrix of A is:

$$C := A^T A \in \mathbb{R}^{\ell \times \ell}.$$

All covariance matrices C are symmetric and positive semi-definite. Thus, all their eigenvalues are non-negative real numbers.
It follows from the Spectral Decomposition Theorem that $\mathbb{R}^\ell$ has an orthonormal basis consisting of eigenvectors of C.

Theorem: PCA via Spectral Decomposition of the Covariance Matrix

Let $A \in \mathbb{R}^{n \times \ell}$ be a data matrix and $C = A^T A$ the corresponding covariance matrix. If the following prerequisites are met, U is a PCA-transformation of A:

- $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_\ell$ are the eigenvalues of C
- $u_1, ..., u_\ell$ is an orthonormal basis of $\mathbb{R}^\ell$, such that u_j is an eigenvector of C associated with λ_j.
- $U \in \mathbb{R}^{\ell \times \ell}$ is the matrix with columns $u_1, ..., u_\ell$.

5.6 Spectral Clustering

The objective of a clustering algorithm is to partition the data into clusters in such a way that

1. the points within each cluster are similar
2. the points in distinct clusters are dissimilar

The k-means clustering algorithm focuses on goal 1 and the spectral clustering algorithm focuses on goal 2. This is useful in situations like they are described here.
TL;DR: Spectral clustering should be used in situations where the points within each cluster are not particularly close together, but the clusters are well-separated (for example, they may be stretched along a line or a circle).

Definition: Similarity Measure

Let X be a set of data points. A similarity measure is a symmetric function

$$s : X \times X \to \mathbb{R}_{\geq 0}$$

Definition: Similarity Matrix

Let s be a similarity measure and $n = |X|$. Then, a similarity matrix is a matrix $S \in \mathbb{R}^{n \times n}$ with $S_{ij} := s(i, j)$.

The objective for spectral clustering is to partition X into k non-empty clusters $C^1, ..., C^k$ in a way that minimizes the overall similarity between points in distinct clusters.

Definition: Minimum Cut

Given a similarity matrix S and a partition $C^1, ..., C^k$, the minimum cut is defined as

$$\text{mincut}(C^1, ..., C^k) := \sum_{p=1}^{k} \sum_{i \in C^p, j \notin C^p} S_{i,j}$$

Definition: Balanced Cut

Given a similarity matrix S and a partition $C^1, ..., C^k$, the balanced cut is defined as

$$\text{balcut}(C^1, ..., C^k) := \sum_{p=1}^{k} \frac{1}{|C^p|} \cdot \sum_{i \in C^p, j \notin C^p} S_{i,j}$$

The clustering that results from minimizing the *minimum cut* favors very small or very large clusters (often obtained by choosing all clusters but one of size 1). Thus, the *balanced cut* is used instead. The only remaining drawback is that minimizing the balanced cut is computationally hard.

Definition: Positive Semi-Definite Matrices

A matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite (p.s.d.) if all its eigenvalues are non-negative numbers.

Definition: The Laplacian

Let S be a similarity matrix.
The Laplacian L of S is the matrix

$$L = D - S \in \mathbb{R}^{n \times n}$$

where D is the diagonal matrix with entries $D_{i,i} = \sum_{j=1}^{n} S_{i,j}$.
The Laplacian is always p.s.d.

Theorem: Connection with Cuts

Let $C^1, ..., C^k$ be a partition of $[n]$ and let $U \in \mathbb{R}^{n \times k}$ be the matrix with entries

$$U_{i,p} = \begin{cases} \frac{1}{\sqrt{|C^p|}} & \text{if } i \in C^p \\ 0 & \text{otherwise} \end{cases}$$

Then $\text{balcut}(C^1, ..., C^k) = \text{trace}(U^T L U)$.

5.6.1 Generalization of the Spectral Clustering

Definition: Partition Matrix

A matrix $U \in \mathbb{R}^{n \times k}$ is a partition matrix if U has exactly k distinct rows.
$C^1, ..., C^k$ is the partition associated with U if each part corresponds to exactly one set of equal rows.

Theorem:

Let $U \in \mathbb{R}^{n \times k}$ be an orthogonal partition matrix and let $C^1, ..., C^k$ be the partition of $[n]$ associated with U. Then

$$\text{balcut}(C^1, .., C^k) = \text{trace}(U^T L U).$$

Thus, finding a partition $C^1, ..., C^k$ that minimizes $\text{balcut}(C^1, ..., C^k)$ is equivalent to finding an orthonormal partition matrix $U \in \mathbb{R}^{n \times k}$ that minimizes $\text{trace}(U^T L U)$.

5.6.2 Relaxation of the Spectral Clustering

The idea is to compute an arbitrary orthonormal U instead of a partition matrix (feasible). Then, find a partition matrix close to the orthonormal matrix U and return the associated partition.

Theorem:

Let $U \in \mathbb{R}^{n \times k}$ be an orthonormal matrix whose columns are eigenvectors to the k smallest eigenvalues of L. Then

$$trace(U^T L U) \leq trace(V^T L V)$$

for all orthonormal matrices $V \in \mathbb{R}^{n \times k}$.

Algorithm: Spectral Clustering

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters
Output: Clusters $C^1, ..., C^k$

1: Compute Laplacian L of S
2: Compute orthonormal matrix $U \in \mathbb{R}^{n \times k}$ whose columns are eigenvectors of the k smallest eigenvalues of L
3: Let $x_1, ..., x_n \in \mathbb{R}^k$ be the rows of U
4: Cluster $x_1, ..., x_n$ using k-means yielding clusters $C^1, ..., C^k$

Intuitively, if the vectors in each of the clusters C^p are close together then we can find a partition matrix $\widehat{U}$ with an associated partition $C^1, ..., C^k$ such that $\widehat{U}$ is close to U and following $\widehat{U}^T L \widehat{U}$ is close to $U^T L T$.

6 Markov Chains

Definition: Transition Matrix

The transition matrix of a Markov chain $\mathcal{Q}$ is a *stochastic* matrix $Q \in \mathbb{R}^{n \times n}$, where q_{ij} is the probability of $\mathcal{Q}$ going from state i to state j.

Definition: Graph of a Markov Chain

The graph of a Markov chain $\mathcal{Q}$ is

$$G_Q = ([n], \{ (i,j) \mid q_{ij} > 0 \}).$$

$\mathcal{Q}$ is connected if G_Q is strongly connected.

Definition: Probability distributions over Markov chains

The *initial probability distribution* of a Markov chain is described by $\mathbf{p}_0$. The probability distribution after t steps is $\mathbf{p}_t = \mathbf{p}_0 Q^t$. The *average probability distribution* after t steps is $\mathbf{a}_t = \frac{1}{t} \sum_{s=1}^{t} \mathbf{p}_s$.

Theorem: Fundamental Theorem of Markov Chains

For every *connected* Markov chain $\mathcal{Q}$ there is a unique vector, called stationary distribution, $\boldsymbol{\pi} \in \mathbb{R}^{1 \times n}$ such that $\boldsymbol{\pi} Q = \boldsymbol{\pi}$.
Moreover, for every initial distribution $\mathbf{p}_0 \in \mathbb{R}^{1 \times n}$:

$$\lim_{t \to \infty} \mathbf{a}_t = \boldsymbol{\pi}$$

Theorem: Characteristic of transition matrices

If Q is the transition matrix of a connected Markov chain, Q has the spectral radius 1.

Definition: Aperiodic and Ergodic

A Markov chain $\mathcal{Q}$ is *aperiodic* if the greatest common divisor of the length of all cycles in G_Q is 1.
A Markov chain is *ergodic* if it is connected and aperiodic.

Theorem: Ergodic Markov Chains

For an ergodic Markov Chain $\mathcal{Q}$, the stationary distribution (resulting from an arbitrary starting distribution $\mathbf{p}_0$) is

$$\lim_{t\to\infty} \mathbf{p}_t = \boldsymbol{\pi}$$

We can convert every Markov chain into an ergodic one by using the following theorem.

Theorem: Ergodic Converstion

Let I be the $(n \times n)$ identity matrix and let $\mathcal{Q}$ be a connected Markov chain and $0 < \alpha < 1$. A Markov chain with transition matrix

$$\alpha Q + (1-\alpha)\cdot I$$

has the same stationary distribution as $\mathcal{Q}$ and is ergodic.

6.1 Markov Chain Monte Carlo Method

We want to sample from a probability space $(\mathbb{U}, \mathcal{P})$, but we only know $\mathcal{P}$ up to an unknown constant Z as $\mathcal{D} = Z \cdot \mathcal{P}$. The idea is to design a Markov chain over $\mathbb{U}$ with stationary distribution $\mathcal{P}$. This can be done by either Metropolis-Hastings or Gibbs sampling.

Definition: Metropolis-Hastings Sampling

Let $(\mathbb{U}, \mathcal{D})$ be a probability space with $\mathcal{G}$ a connected undirected graph with maximum degree Δ and $V(\mathcal{G}) = \mathbb{U}$.
Then, $\mathcal{Q}$ is the Metropolis-Hastings Markov Chain if it has transition matrix:

$$q_{uv} = \begin{cases} \frac{1}{\Delta} & \text{if } uv \in E(\mathcal{G}) \text{ and } \mathcal{D}(v) \geq \mathcal{D}(u) \\ \frac{1}{\Delta} \cdot \frac{\mathcal{D}(v)}{\mathcal{D}(u)} & \text{if } uv \in E(\mathcal{G}) \text{ and} \mathcal{D}(v) < \mathcal{D}(u) \\ 1 - \sum_{v' \in N(u)} q_{uv'} & \text{if } u = v \\ 0 & \text{otherwise} \end{cases}$$

Algorithm: Metropolis-Hastings Sampling

Input: Probability space $(\mathbb{U}, \mathcal{D})$, connected undirected graph $\mathcal{G}$ with maximum degree Δ and $V(\mathcal{G}) = \mathbb{U}$, $u \in \mathbb{U}$ current state

Output: $v \in \mathbb{U}$ next state sampled from the Metropolis-Hastings Markov Chain

---

1: $b \leftarrow \begin{cases} 1 & \text{with probability } \frac{|N(u)|}{\Delta} \\ 0 & \text{otherwise} \end{cases}$
2: **if** $b = 1$ **then**
3: choose a neighbour $v' \in N(u)$ in $\mathcal{G}$ uniformly at random
4: **if** $\mathcal{D}(v') \geq \mathcal{D}(u)$ **then**
5: $v \leftarrow v'$
6: **else**
7: $v \leftarrow \begin{cases} v' & \text{with probability } \mathcal{D}(v')/\mathcal{D}(u) \\ u & \text{with probability } 1 - \mathcal{D}(v')/\mathcal{D}(u) \end{cases}$
8: **end if**
9: **else**
10: $v \leftarrow u$
11: **end if**
12: **return** v

Theorem: Metropolos-Hasings Sampling

The stationary distribution of the Metropolis-Hastings Markov Chain is $\mathcal{P}$.

Definition: Gibbs Sampling

Let $(\mathbb{U} = \mathbb{D}^l, \mathcal{D})$ be a probability space with $\mathbb{D}$ finite.

Then, $\mathcal{Q}$ is the Gibbs Markov Chain if it has transition matrix:

$$q_{uv} = \begin{cases} \frac{1}{l} \sum_{i=1}^{l} \mathcal{P}(u_i | u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_l) & \text{if } u = v \\ \frac{1}{l} \mathcal{P}(v_i | u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_l) & \text{if } u \text{ and } v \text{ differ in exaclty one } i \in [l] \\ 0 & \text{otherwise} \end{cases}$$

Algorithm: Gibbs Sampling

Input: Probability Space $(\mathbb{U} = \mathbb{D}^l, \mathcal{D})$ with $\mathbb{D}$ finite, $u \in \mathbb{U}$ the current state.

Output: Next state $v \in \mathbb{U}$ sampled from the Gibbs Markov Chain

---

1. Choose $i \in [l]$ uniformly at random
2. Compute $Z_{u,i} = \sum_{v \in \mathbb{D}} \mathcal{D}(u_1, \ldots, u_{i-1}, v, u_{i+1}, \ldots, u_l)$
3. Choose $v_i \in \mathbb{D}$ with probability

$$\frac{\mathcal{D}(u_1, \ldots, u_{i-1}, v, u_{i+1}, \ldots, u_l)}{Z_{u,i}}$$

4. Return $v = (u_1, \ldots, u_{i-1}, v_i, u_{i+1}, \ldots, u_l)$

Theorem: Gibbs Sampling

The stationary distribution of the Gibbs Markov chain is $\mathcal{P}$.

6.1.1 Bounding the Mixing Time

We need to make sure that the Markov chain converges quickly to the stationary distribution. That is, we need to bound the convergence rate (i.e. *mixing time*) of the chain.
This is usually difficult, and in fact, many Markov chains converge very slowly to their stationary distribution. But there are cases in which it is possible to prove a fast convergence.

Definition: Total Variation Distance

The total variation distance between two probability distributions $\mathcal{P}, \mathcal{P}'$ over the same state space $\mathbb{U}$ is

$$\|\mathcal{P} - \mathcal{P}'\|_{TV} := \frac{1}{2} \sum_{u \in \mathbb{U}} |\mathcal{P}(u) - \mathcal{P}'(u)| = \max_{A \subseteq \mathbb{U}} |\mathcal{P}(A) - \mathcal{P}'(A)|$$

Theorem: Jerrum and Sinclair

The mixing time of the Markov chain for uniformly sampling matchings of a graph G is polynomial in the size of G.

6.2 Page Rank

A search engine has two tasks:

1. Find the set of web pages containing the query term.
2. Rank the web pages and return them in a ranked order.

This section will discuss how task 2 can be completed.

6.2.1 Equivalent Basic Ideas of Page Rank

Idea 1:
A web page is important if many links point to it. However, links coming from important web pages should carry higher weights.

Idea 2:
If we take a random walk on the web graph, the pages we visit more often are more important.

6.2.2 Web Graph as Markov Chain

The web graph G_{web} is defined as follows. The web pages are numbered $1, ..., n$. Let $E(G_i) := \{(i, j) \mid \text{page } i \text{ has link to page } j\}$.
Let Q_{web} be the Markov chain of the random walk on G_{web} where each edge has the same weight. The transition matrix is then $Q_{\text{web}} = q_{ij} \in \mathbb{R}^{n \times n}$ with

$$q_{ij} = \begin{cases} \frac{1}{deg_+(i)} & \text{if } (i, j) \in E(G_{\text{web}}) \\ 0 & \text{otherwise} \end{cases}$$

6.2.3 Implementation of the Page Rank

Use a non-negative weight vector $w = (w_1, ..., w_n) \in \mathbb{R}^{1\times n}$ where w_i measures the importance of page i. We normalize the vector so that $\sum_i w_i = 1$.
At the beginning, all pages have the same weight $w_i = \frac{1}{n}$. The goal is to assign more weight to pages with higher in-degree. To do this we repeatedly update w as follows

$$w \leftarrow wQ_{\text{web}}$$

The resulting sequence $w_0, w_1, ...$ is exactly the sequence p_0, p_1 of probability distributions on the Markov Chain Q_{web} when it is started with the initial distribution $p_0 = w_0 = (\frac{1}{n}, ..., \frac{1}{n})$.
If the Markov chain Q_{web} is ergodic this sequence converges to the stationary distribution π. In practice, however, Q_{web} is not ergodic because the web graph is not connected. In this case, a technique called *random restarts* is used. At any point with a small probability, the chain jumps to an arbitrary vertex instead of following an edge.

7 Algorithms for Massively Parallel Systems

7.1 Map-Reduce Programming Model

The Map-Reduce model is a programming model for data analysis on a massively parallel system. A Map-Reduce program computation consists of three phases:

Map: A Map function receives one key-value pair and emits zero or more key-value pairs. The function is executed for every input key-value pair in parallel on the system.

Shuffle: All emitted key-value pairs are sorted by their key component.

Reduce: A Reduce function receives all key-value pairs for one key and emits zero or more key-value pairs. The function is executed for every key-value in parallel on the system.

The user has to provide the Map and Reduce function. The system takes care of the rest. Multiple computations can be chained together, resulting in a Map-Reduce process.

7.2 Relational Algebra in Map-Reduce

For a given relation $\mathcal{R}$ with schema $R(A_1, \ldots, A_l)$ a tuple t is stored as akey-value pair (R, t).

The fundamental operations of relational algebra can easily be implemented as Map-Reduce programs. The implementations are given for exemplary cases, i.e. input schemata and operation parameters.

Algorithm: Projection

Input: Relation $\mathcal{R}$ with schema $R(A, B, C)$

Output: $\mathcal{Q} = \pi_{A,C}(\mathcal{R})$

Map: On input $(R, (a, b, c))$ emit $((a, c), 1)$.

Reduce: On input $((a, c), \text{values})$ emit $(Q, (a, c))$.

Algorithm: Intersection

Input: Relations $\mathcal{R}$ and $\mathcal{S}$ with the same attributes

Output: $\mathcal{Q} = \mathcal{R} \cap \mathcal{S}$

MAP: On input (R, t) emit (t, R), on input (S, t) emit (t, S).

REDUCE: On input (t, values) emit (Q, t), if values contains R and S.

Algorithm: Join

Input: Relation $\mathcal{R}$ with schema $R(A, B)$ and relation $\mathcal{S}$ with schema $S(B, C)$

Output: $\mathcal{Q} = \mathcal{R} \bowtie \mathcal{S}$

MAP: On input $(R, (a, b))$ emit $(b, (R, a))$, on input $(S, (b, c))$ emit $(b, (S, c))$.

REDUCE: On input (b, values) emit $(Q, (a, b, c))$ for all $(R, a), (S, c) \in \text{values}$.

Algorithm: Grouping and Aggregation

Input: Relation $\mathcal{R}$ with schema $R(A, B, C)$

Output: Relation $\mathcal{Q}$ resulting by grouping $\mathcal{R}$ by attribute A and take the average over attribute C

MAP: On input $(R, (a, b, c))$ emit (a, c).

REDUCE: On input (a, values) compute the average $c^\star$ of the entries $c \in \text{values}$ and emit $(Q, (a, c^\star))$.

7.3 Matrix Multiplication

First, we take a look at matrix-vector multiplication. The first variant is used when vector v fits into main memory. The second variant is used when v exceeds the main memory.

Algorithm: Matrix-Vector Multiplication (1)

Input: $v \in \mathbb{R}^n$ $A \in \mathbb{R}^{m \times n}$ stored as:
- v in the main memory of each worker
- $((i, j), a_{ij}))$

Output: $Av \in \mathbb{R}^n$

MAP: On input $((i, j), a_{ij}))$ emit $(i, a_{ij} v_j)$.

REDUCE: On input (i, values) emit $(i, \sum_{v \in \text{values}} v)$.

The second variant for larger than main memory vectors v works with a partition of $[n]$, which is used to partition v into segments and A into vertical stripes. Each MAP worker should only get key-value pairs that are in the same stripe, so it can keep the same segment of v loaded.

Algorithm: Matrix-Vector Multiplication (2)

Input: $v \in \mathbb{R}^n$ $A \in \mathbb{R}^{m \times n}$ stored as:
- v partitioned into k segments
- $((i,j), a_{ij}))$ partitioned into vertical stripes

Output: $Av \in \mathbb{R}^n$

Map: On input $((i,j), a_{ij}))$ if a_{ij} is in stripe k, load section k of v into memory and emit $(i, a_{ij} v_j)$.

Reduce: On input (i, values) emit $(i, \sum_{v \in \text{values}} v)$.

Algorithm: Two Round Matrix Multiplication

Input: $A \in \mathbb{R}^{\ell \times m}, B \in \mathbb{R}^{m \times n}$ stored as:
- $(A, (i,j,a_{ij}))$
- $(B, (j,k,b_{jk}))$

Output: $C = AB \in \mathbb{R}^{\ell \times n}$

First Map: On input $(A, (i,j,v))$ emit $(j, (A,i,v))$ on input $(B, (j,k,w))$ emit $(j, (B,k,w))$.

First Reduce: 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: The identity function: on input $((i,k), x)$ emit $((i,k), x)$.

Second Reduce: On input $((i,k), \text{values})$ compute the sum $x^\star$ of all $x \in \text{values}$ and emit $(C, (i,k,x^\star))$.

The idea of the two round algorithm is to look at matrix multiplication from a relational algebra perspective. Both matrices are stored as ternary relations. The first Map-Reduce round joins both relations and simultaneously computes all multiplications. The second Map-Reduce round groups all products of the first round by their position in the output matrix and computes a sum.

Algorithm: One Round Matrix Multiplication

Input: $A \in \mathbb{R}^{\ell \times m}, B \in \mathbb{R}^{m \times n}$ stored as:
- $(A, (i,j,a_{ij}))$
- $(B, (j,k,b_{jk}))$

Output: $C = AB \in \mathbb{R}^{\ell \times n}$

Map: On input $(A, (i,j,v))$ emit $((i,k), (A,j,v))$ for $k \in [n]$, on input $(B, (j,k,w))$ emit $((i,k), (B,j,w))$ for $i \in [l]$.

Reduce: On input $((i,k), \text{values})$ compute the sum x of all vw for $(A,j,v), (B,j,w) \in \text{values}$ and emit $(C, (i,k,x))$.

The one round algorithm is a clever way to compute the join in one Map phase and the grouping and sum in one reduce phase.

7.4 Analysis of Map-Reduce Algorithms

Definition: Cost Measures

- **Wall-clock time:** Total time for the MR-process to finish.
- **Number of rounds:** Number of MR-rounds in the process.
- **Communication cost:** Sum of input sizes to all phases.
- **Replication rate:** Number of key-value pairs produced by all map tasks divided by the input size.
- **Maximum load:** Maximum input length of reduce tasks.

Wall-clock time: This is the ultimate parameter we are interested in, but it is heavily system-depended and requires complicated analysis.

Number of rounds: This number is a reasonable and important cost factor but has to be viewed in conjunction with other measures to be meaningful.

Communication cost: In practical settings with large amounts of data the execution cost is dominated by the cost of transferring the data. Since each output (except for the final one which is small) is input to the next task, outputs can be ignored.
The input size can be measured in bits or more abstractly such as number of tuples.
The measure only works for algorithms that balance the load "reasonably". A MR process that puts all computations into one node would minimize the communication cost but is of course pointless.

Replication rate: This measure puts the communication cost into perspective and only works for single-round MR-processes.

Maximum load: Measures load balancing and has an impact on the execution time of reducers.

7.4.1 Analysis of Matrix Multiplication

We assume the non-zero entries of the two matrices are randomly distributed. Formally:

- $P(a_{ij} \neq 0) = p$ independently for all i, j.
- $P(b_{jk} \neq 0) = q$ independently for all j, k.

for (small) p, q with $0 \leq p, q \leq 1$.

Theorem: Analysis of the two-round algorithm

- Expected communication cost: $2plm + 2qmn + 2pqlmn$.
- Maximum load in the first round: $pl + qn$, with high probability below $(1 + \varepsilon)(pl + qn)$.
- Maximum load in the second round: pqm, with high probability below $(1 + \varepsilon)(pqm)$.

Theorem: Analysis of the one-round algorithm

- Expected communication cost: $plm + qmn + (p + q)lmn$.
- Maximum load: $pm + qm$, with high probability below $(1 + \varepsilon)(pm + qm)$.

We generalize the single round matrix multiplication algorithm by dividing the matrices in s stripes. We define a mapping $h : [n] \to [s]$ that assigns each column/row of a matrix to a stripe.

Algorithm: Generalized Single Round Algorithm

Input: $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times n}$ stored as:
- $(A, (i, j, a_{ij}))$
- $(B, (j, k, b_{jk}))$

Output: $C = AB \in \mathbb{R}^{n \times n}$

Map: On input $(A, (i, j, v))$ emit $((h(i), u), (A, i, j, v))$ for $u \in [s]$, on input $(B, (j, k, w))$ emit $((t, h(k)), (B, j, k, w))$ for $t \in [s]$.

Reduce: On input $((t, u), \text{values})$ for all $i \in h^{-1}(t)$ and $k \in h^{-1}(u)$ compute the sum c_{ik} of all vw for $(A, i, j, v), (B, j, k, w) \in \text{values}$ and emit $(C, (i, k, c_{ik}))$.

Theorem: Analysis of the generalized single round algorithm

Worst case:
- Replication rate: s.
- Communication cost: $2n^2 + 2sn^2 = \mathcal{O}(sn^2)$.
- Each reducer gets all entries of n/s rows and n/s columns. Thus the maximum load is $\frac{2n^2}{s}$.

Average case:
- Replication rate: s.
- Communication cost: $(p+q)n^2 + s(p+q)n^2 = \mathcal{O}(s(p+q)n^2)$.
- Each reducer gets all non-zero entries of n/s rows and n/s columns. Thus the maximum load is $\frac{(p+q)n^2}{s}$.

7.5 Multiway Joins in Map Reduce

The task is to compute the natural join of multiple relations.

Definition: Multiway Natural Join

Input: $\mathcal{R}_1, \ldots, \mathcal{R}_l$ with:

Output: $\mathcal{Q} = \mathcal{R}_1 \bowtie \ldots \bowtie \mathcal{R}_l$

Further parameters:
- $R_i(A_{i1}, \ldots, A_{ik_i})$ schema of $\mathcal{R}_i$
- $\{ A_1, \ldots, A_k \}$ set of all attributes
- V_j domain of attribute A_j

The Hypercube algorithm has the following parameters:
- $s_j \in \mathbb{N}$ share of the attribute A_j
- $\prod_{j=1}^{k} s_j = s$ number of reducers
- $h_1, \ldots, h_k$ independently chosen hash functions $h_j : V_j \to [s_j]$

Algorithm: MAP function of Hypercube

Input: $(R_i, (a_1, \ldots, a_{k_i}))$
Output: $(\overline{p}, \text{values})$

On input $(R_i, (a_1, \ldots, a_{k_i}))$, emit

$$((p_1, ..., p_k), (R_i, (a_1, ..., a_{k_i})))$$

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 $A_{ij'} = A_j$

Algorithm: REDUCE function of Hypercube

Input: $(\overline{p}, \text{values})$
Output: $(Q_i, (a_1, \ldots, a_k))$

Compute

$$\mathcal{Q}(\overline{p} := \mathcal{R}_1(\overline{p}) \bowtie \ldots \bowtie \mathcal{R}_\ell(\overline{p})$$

where

$$\mathcal{R}_i(\overline{p}) := \{t \mid (R_i, t) \in \text{values}\}$$

and emit all pairs (Q, t) for $t \in \mathcal{Q}(\overline{p})$

Theorem: Analysis of the Hypercube Algorithm

For every $i \in [1, \ell]$ we have $Idx(i) := \{j \in [1, k] \mid A_j \in \{A_{i1}, ..., A_{ik_i}\}\}$ and $m_i := |\mathcal{R}_i| \forall i \in [1, \ell]$.

1. The replication rate of the Hypercube algorithm is:

$$\frac{\sum_{i=1}^{\ell} \left(m_i \cdot \prod_{j \in [k] \setminus \text{Idx}(i)} s_j \right)}{\sum_{i=1}^{\ell} m_i}$$

2. The expected load of the algorithm (over the random choices of the hash functions) is

$$\sum_{i \in [l]} \frac{m_i}{\prod_{j \in \text{Idx}(i)} s_j}$$

3. With a high probability, the maximum load is

$$\mathcal{O}\left(\sum_{i \in [\ell]} \frac{m_i}{\min_{j \in Idx(i)} s_j} \right)$$

The base idea is to divide each attribute into shares using the hash functions (similar to the stripes in matrix multiplication) and let the reducer perform the join on a small subset according to these shares. The number of shares per attribute is a vital parameter and should be chosen in a way that minimizes the expected load.

7.5.1 Skew-Free Relations

Skew-free relations have a special property regarding the maximum load.

Definition: Frequency of a Tuple

Let $i \in [\ell]$ and $J \subseteq Idx(i)$. 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.

Definition: Skew-Free Relation

A relation $\mathcal{R}_i$ is skew-free with respect to $s_1, ..., s_k$, if for every set $J \subseteq Idx(i)$ and for 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: Maximum Load of Skew-Free Relations

If the relations $\mathcal{R}_1, ..., \mathcal{R}_\ell$ are skew-free with respect to $s_1, ..., s_k$ then with a high probability the maximum load is

$$\mathcal{O}\left(\sum_{i \in [\ell]} \frac{m_i}{\prod_{j \in Idx(i)} s_j} (\log(s))^k\right)$$

8 Streaming Algorithms

Sometimes the amount of data is too much to store it or there is no need to store the data after processing. Therefore, the goal is to design efficient (sublinear space, online, real-time) algorithms for data analysis tasks.

The formal model for this setting is defined as follows. The data items are from an *universe* $\mathbb{U}$ with $N := |\mathbb{U}|$. Sometimes the assumption $\mathbb{U} = \{0, \ldots, N-1\}$ is made. $a = a_1, ..., a_n$ is the *input stream* with $a_i \in \mathbb{U}$.
The length n of the data stream is not known to the algorithm in advance. The most interesting property is the space usage of the algorithm. Typically, an algorithm should have the following space usage:

$$\texttt{polylog}(n+N) = \bigcup_{k \geq 1} \log(n+N)^k$$

where the assumption $n + N \geq 2$ is made to avoid edge cases.

8.1 Sampling from Streams

The task is to pick elements a_i uniformly at random from the elements of the stream. This is trivial if n is known in advance. But if n is unknown, the following algorithm is used.

Algorithm: Simple Sampling Algorithm

Input: Stream $a_1, ..., a_n$
Output: A uniformly picked element from the stream.

1: $i \leftarrow 0$
2: **while** not end of stream **do**
3: $\quad i \leftarrow i + 1$
4: $\quad$ sample $\leftarrow a_i$ with probability $\frac{1}{i}$
5: **end while**
6: **return** sample

This time, we want to sample k elements uniformly. Note that there are $\binom{n}{k}$ possible reservoirs.

Algorithm: Simple Sampling Algorithm

Input: Stream $a_1, ..., a_n$ and $k \leq n$
Output: k uniformly picked elements from the stream.

1: **for** $i = 1, ..., k$ **do**
2: $\quad$ sample[i]$\leftarrow a_i$
3: **end for**
4: **while** not end of stream **do**
5: $\quad i \leftarrow i + 1$
6: $\quad$ replace $\leftarrow \begin{cases} true & \text{with probability } \frac{k}{i} \\ false & \text{otherwise} \end{cases}$
7: $\quad$ **if** replace **then**
8: $\quad\quad$ choose j uniformly at random from $[k]$
9: $\quad\quad$ sample[j]$\leftarrow a_i$
10: $\quad$ **end if**
11: **end while**
12: **return** sample

The space complexity of the algorithm is $\mathcal{O}(\log n + k \cdot \log N)$.

8.2 Hash Functions

Definition: Hash Function

A hash function h on a universe $\mathbb{U}$ is a function from $\mathbb{U}$ to a set $\mathbb{T}$ (usually an initial segment of the natural numbers). h is assumed to be or at least look random.
Formally, consider a probability distribution on the space of all functions $h : \mathbb{U} \to \mathbb{T}$. If it is uniform, the hash function is *truly random*.

Most analyses of algorithms based on hashing assume 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.

8.2.1 True Randomness

Suppose we choose h uniformly at random from the class $\mathbb{T}^{\mathbb{U}}$ of all functions from $\mathbb{U}$ to $\mathbb{T}$ ($|\mathbb{T}| = M$). The following properties show the use of true randomness:

- 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 $x_1, ..., x_k \in \mathbb{U}$ and all $y_1, ..., y_k \in \mathbb{T}$ we have

$$\Pr_{h \in \mathbb{T}^{\mathbb{U}}}(h(x_1) = y_1 \wedge ... \wedge h(x_k) = y_k) = \frac{1}{M^k}$$

Since randomness is hard to obtain, true randomness is unrealistic. Generating a random function from $\{0, ..., N-1\} \to \{0, ..., M-1\}$ requires $\Theta(N \cdot \log M)$ random bits. Storing it needs also $\Theta(N \cdot \log M)$ bits. Since N is typically very large, the space requirement alone is prohibitive.

8.2.2 Families of Hash Functions

By fixing a small family $\mathcal{H}$ of hash functions from $\mathbb{U}$ to $\mathbb{T}$ and considering the uniform distribution on this family one can obtain feasible distributions of hash functions.

Definition: Universal Hashing

A family $\mathcal{H}$ of hash functions from $\mathbb{U}$ to $\mathbb{T}$ is universal if for all distinct $x, x' \in \mathbb{U}$

$$\Pr_{h \in \mathcal{H}}(h(x) = h(x')) \leq \frac{1}{|\mathbb{T}|}$$

8.2.3 Signatures

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.

Definition: Number of Collisions

For a function $h : \mathbb{U} \to \mathbb{T}$ and a set $S \subseteq \mathbb{U}$ we let

$$coll(h, S) := |\{\{x, x'\} \mid x, x' \in S \text{ such that } x \neq x' \text{ and } h(x) = h(x')\}|$$

denote the number of collisions of h on S.

Theorem: Expected Number of Collisions

Let $\mathcal{H}$ be a universal family of hash functions from $\mathbb{U}$ to $\{0, ..., 2^k - 1\}$. Then for every $\delta > 0$ and every set $S \subseteq \mathbb{U}$ of cardinality $|S| = n$,

$$E_{h \in \mathcal{H}}(coll(h, s)) = \frac{n(n-1)}{2^{k+1}} \quad \text{and} \quad \Pr_{h \in \mathcal{H}}\left(coll(h, S) \geq \frac{n^2}{\delta 2^{k+1}}\right) \leq \delta.$$

Theorem:

Let $n \in \mathbb{N}$ and $\delta > 0$. Let $\mathcal{H}$ be a universal family of hash functions from $\mathbb{U}$ to $\{0, ..., 2^k - 1\}$ where $k \geq 2 \log n + \log \frac{1}{\delta} - 1$. Then for every set $S \subseteq \mathbb{U}$ of cardinality $|S| \leq n$,

$$\Pr_{h \in \mathcal{H}}(coll(h, S) \geq 1) \leq \delta.$$

8.2.4 Strongly k-universal Families

Definition: k-universal and strongly k-universal families

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 $x_1, ..., x_k \in \mathbb{U}$

$$\Pr_{h \in \mathcal{H}}(h(x_1) = h(x_2) = ... = h(x_k)) \leq \frac{1}{|\mathbb{T}|^{k-1}}$$

2. $\mathcal{H}$ is strongly k-universal if for all distinct $x_1, ..., x_k \in \mathbb{U}$ and all $y_1, ..., y_k \in \mathbb{T}$

$$\Pr_{h \in \mathcal{H}}(h(x_1) = y_1 \wedge ... \wedge h(x_k) = y_k) = \frac{1}{|\mathbb{T}|^k}$$

Theorem: Alternative Characterization of Strongly k-Universal Families

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 properties.

1. **k-independence:** For all distinct $x_1, ..., x_k \in \mathbb{U}$ and all $y_1, ..., y_k \in \mathbb{T}$

$$\Pr_{h \in \mathcal{H}}\left(\bigwedge_{i=1}^{k} h(x_i) = y_i\right) = \prod_{i=1}^{k} \Pr_{h \in \mathcal{H}}(h(x_i) = y_i).$$

2. **Uniformity:** For all $x \in \mathbb{U}$ and $y \in \mathbb{T}$

$$\Pr_{h \in \mathcal{H}}(h(x) = y) = \frac{1}{|\mathbb{T}|}$$

8.2.5 Construction of Strongly k-Universal Families

The following steps are executed to construct a strongly k-universal family.

1. Choose a prime power $q \geq N$ and let $\mathbb{F}_q$ denote the field with q elements (unique up to isomorphism).
2. Fix an arbitrary injection $g_1 : \mathbb{U} \to \mathbb{F}_1$ and an arbitrary bijection $g_2 : \mathbb{F}_q \to \{0, ..., q-1\}$.
3. For $a = (a_0, ..., a_{k-1}) \in \mathbb{F}_q^k$ let $p_a : \mathbb{F}_q \to \mathbb{F}_q$ be the polynomial function

$$p_a(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1}$$

and let $f_a : \mathbb{U} \to \{0, ..., q-1\}$ be the function $g_2 \circ p_a \circ g_1$.

4. Define functions $h_a : \mathbb{U} \to \{0, ..., M-1\}$ by: $h_a(x) := f_a(x) \mod M$
5. Let $\mathcal{H}^k_{q.M} := \{h_a \mid a \in \mathbb{F}^k_q\}$

Theorem: Special Strongly k-universal Families

The family $\mathcal{H}^k_q := \{f_a \mid a \in \mathbb{F}^k_q\}$ of hash functions from $\mathbb{U}$ to $\{0, ..., q-1\}$ is strongly k-universal.

If M divides q, then the family $\mathcal{H}^k_{q,M}$ is strongly k-universal.

Even if M does not divide q the family $\mathcal{H}^k_{q,M}$ is close to strongly k-universal as long as $M \ll q$.

Theorem: Properties of $\mathcal{H}^k_{q,M}$-families

For all M the family $\mathcal{H}^k_{q,M}$ satisfies the following two conditions.

1. **Independence:** For all distinct $x_1, ..., x_k \in \mathbb{U}$ and all $y_1, ..., y_k \in \{0, ..., M-1\}$

$$\Pr_{h \in \mathcal{H}} \left(\bigwedge_{i=1}^{k} h(x_i) = y_i \right) = \prod_{i=1}^{k} \Pr_{h \in \mathcal{H}} (h(x_i) = y_i).$$

2. **Almost Uniformity:** For $x \in \mathbb{U}$ and all $y \in \{0, ..., M-1\}$

$$\left| \Pr_{h \in \mathcal{H}} (h(x) = y) - \frac{1}{M} \right| \leq \frac{1}{q}.$$

8.3 Counting Distinct Elements

We have a universe $\mathbb{U}$ of size $|\mathbb{U}| = N$ and a data stream $a_1, ..., a_n$ of items from $\mathbb{U}$. The task is to count the number of distinct elements in the stream $a_1, ..., a_n$. While there are obvious solutions using space $O(N)$ or space $O(n \log N)$, this problem cannot be solved exactly in $space < \min\{N, n\}$.

8.3.1 Approximately Counting Distinct Elements

While the linear space lower bound for exact counting is prohibitive for very large N and n, for many applications, it is sufficient to count the number of distinct elements in a data stream approximately.
We assume that the elements $a_1, ..., a_n$ are chosen uniformly at random from $\mathbb{U}$. For $a > 0$ let

$$zeroes(a) = \max\{i \mid 2^i \text{ divides } a\}$$

be the number of trailing zeroes the binary representation of a.

Algorithm: ZCount

Input: Stream a.
Output: Estimator for the number of elements in the stream.

```
z ← 0
while not end of stream do
    a ← next stream element
    if zeroes(a) > z then
        z ← zeroes(a)
    end if
end while
return 2^{z+1/2}
```

8.3.2 The Flajolet-Martin Algorithm

Let $\mathcal{H}$ be a strongly 2-universal family of hash function from $\mathbb{U}$ to $[M]$ where M is the first power of 2 greater than or equal to N.

Algorithm: FMCount

Input: Stream a and 2-universal family of hash functions $\mathcal{H}$.
Output: Estimator for the number of elements in the stream.

Draw h uniformly at random from $\mathcal{H}$
$z \leftarrow 0$
while not end of stream **do**
    $a \leftarrow$ next stream element
    if $zeroes(h(a)) > z$ **then**
        $z \leftarrow zeroes(h(a))$
    end if
end while
return $2^{z+\frac{1}{2}}$

The algorithm needs $\mathcal{O}(\log N)$ memory space.

Theorem: Approximation Guarantee of FMCount

Let $d = d(a_1, ..., a_n)$ be the number of distinct elements in the input stream $d^* = d^*(h, a_1, ..., a_n)$ be the estimator returned by the FMCount algorithm. Then

$$\Pr_{h \in \mathcal{H}}\left(d^* \leq \frac{1}{3}d\right) \leq \frac{\sqrt{2}}{3}$$

$$\Pr_{h \in \mathcal{H}}(d^* \geq 3d) \leq \frac{\sqrt{2}}{3}.$$

This initial confidence bound is not great but can be improved by using the median trick which is implemented as the MCount(k)-algorithm.

Algorithm: MCount(k)

For a $k \geq 1$:

1. Run $2k-1$ copies of FMCount in parallel with hash functions $h_1, ..., h_{2k-1}$ drawn independently from a family of hash functions $\mathcal{H}$.
2. Let $d^1, ..., d^{2k-1}$ be the resulting estimators for the number d of distinct elements in the input stream.
3. Return the median d^* of $d^1, ..., d^{2k-1}$.

Theorem: Approximation Guarantee of MCount(k)

Let $d = d(a_1, ..., a_n)$ be the number of distinct elements in the input stream. For every $\delta > 0$ there exists a $k = \mathcal{O}(ln(\frac{1}{\delta}))$ such that the estimator d^* returned by MCount(k) satisfies

$$Pr\left(\frac{d}{3} < d^* < 3d\right) \geq 1 - \delta$$

The MCount(k) algorithm needs $\mathcal{O}(k \cdot \log N)$ memory space.

8.4 Frequency Moments

Let $a = a_1, .., a_n$ be a data stream consisting of elements from $\mathbb{U}$ and let $u \in \mathbb{U}$.

Definition: Frequency and Frequency Moments

The *frequency* of u in a is

$$f_u(a) := |\{i \in [n] \mid a_i = u\}|.$$

Let $p \geq 0$ be real and non-negative. The pth *frequency moment* of a is

$$F_p(a) := \sum_{u \in \mathbb{U}} (f_u(a))^p.$$

For the case $p = 0$ we restrict the sum to strictly positive f_u.

We can interpret $\sqrt[p]{F_p} := \|f\|_p$ as the L_p-norm of f.

It is possible to compute the average variance using frequencies. Assume the elements from the stream are being chosen uniformly at random. Then the expected frequency is

$$E(f_u) = \sum_{i=1}^{n} \frac{1}{N} = \frac{n}{N}$$

Following from this the average variance is

$$\frac{1}{N} \sum_{u \in \mathbb{U}} E(f_u - E(f_u - E(f_u))^2 = \frac{1}{N} F_2 - \frac{n^2}{N^2}$$

8.4.1 An Estimator for F_k

Let $k \in$ with $k \geq 2$ and let $a = a_1, ..., a_n$ be the input stream. Then the estimator A_k is picked as follows.

1. Pick an index $i \in [n]$ uniformly at random.
2. Let $r := |\{j \geq i \mid a_j = a_i\}|$.
3. Let $A_k := n(r^k - (r-1)^k)$.

Then, $E(A_k) = F_k$

Algorithm: AMS-Estimator

Input: Stream a and $k \in \mathbb{N}$.
Output: Estimator for F_k.

```
1:  i = 0
2:  while not end of stream do
3:      i ← i + 1
4:      with probabiliy 1/i do
5:          a ← a_i
6:          r ← 0
7:      if a_i = a then
8:          r ← r + 1
9:      end if
10: end while
11: return i(r^k − (r − 1)^k)
```

8.4.2 An Estimator for F_2

To estimate F_2, we use the Tug-of-War algorithm. Let $\mathcal{H}$ be a strongly 4-universal family of hash functions from $\mathbb{U}$ to $\{-1, 1\}$.

Algorithm: Tug-of-War

Input: Strongly 4-universal hash family $\mathcal{H}$.
Output: Estimator for F_2.

```
draw h uniformly at random from H
x ← 0
while not end of stream do
    a ← next element from stream
    x ← x + h(a)
end while
return x^2
```

Let B be the estimator returned by the Tug-of-War algorithm. Then

$$E(B) = F_2 \text{ and } Var(B) \leq 2 \cdot F_2^2.$$

There is a variation of the Tug-of-War algorithm called Avg-ToW(k).

Algorithm: Avg-ToW(k)

Input: Strongly 4-universal hash family $\mathcal{H}$.
Output: Estimator for F_2.

```
draw h_1, ..., h_k independently from H
for i = 1, ...k do
    x_i ← 0
end for
while not end of stream do
    a ← next element from stream
    for i = 1, ..., k do
        x_i ← x_i + h_i(a)
    end for
end while
return (1/k) Σ_{i=1}^{k} x_i^2
```

Theorem: Precision of Avg-ToW(k)

Let $\epsilon, \delta > 0$, $k = \lceil \frac{2}{\epsilon^2 \delta} \rceil$ and let B be the estimator returned by Avg-ToW(k). Then $E(B) = F_2$ and

$$Pr(|B - F_2| < \epsilon F_2) > 1 - \delta.$$

8.5 Sketching

In our current setting, data is stored in a long vector that we want to query. A data stream contains a sequence of updates to the vector. Our goal is to obtain a *sketch* of the vector, i.e. a space-efficient summary that enables us to answer queries approximately and allows for efficient updates.

Definition: Turnstile Data Stream Model

Given a universe $\mathbb{U}$ of size N and a stream of updates $(a_1, c_1), ..., (a_n, c_n)$ with $a_i \in \mathbb{U}, c_i \in \mathbb{Z}$, we have a *data vector* $d := d(n)$ defined as

$$d(i) = (d_u(i))_{u \in \mathbb{U}} \in \mathbb{Z}^{\mathbb{U}} \quad \text{for } 0 \leq i \leq n$$

defined by

$$d_u(0) := 0$$

$$d_u(i+1) := \begin{cases} d_u(i) + c_i & \text{if } u = a_i \\ d_u(i) & \text{if } u \neq a_i \end{cases}$$

For the following algorithms, we need to impose additional restrictions on the data model

Definition: Strict Turnstile Model

All entries of the data vector are nonnegative at any time, i.e. $d_u(i) \geq 0$ for all $u \in \mathbb{U}$, $i \in [n]$

Definition: Cash Register Model

All updates are positive, i.e. $c_i > 0$ for all $i \in [n]$

Algorithm: Simple Sketch(k)

Input: Universal hash family $\mathcal{H}$ from $\mathbb{U}$ to $[k]$.
Output: Sketch S.

draw h from $\mathcal{H}$
for $i = 1, ..., k$ **do**
    $S[i] := 0$
end for
while not end of stream **do**
    $(a, c) \leftarrow$ next update
    $S[h(a)] \leftarrow S[h(a)] + c$
end while
return S

Algorithm: Count Min Sketch(k, ℓ)

Input: Universal hash family $\mathcal{H}$ from $\mathbb{U}$ to $[k]$.
Output: Sketch S.

draw $h_1, ..., h_\ell$ independently from $\mathcal{H}$
for $i = 1, ..., k$ **do**
    for $j = 1, ..., \ell$ **do**
        $S[i, j] \leftarrow 0$
    end for
end for
while not end of stream **do**
    $(a, c) \leftarrow$ next update
    for $j = 1, ..., \ell$ **do**
        $S[h_j(a), j] \leftarrow S[h_j(a), j] + c$
    end for
end while
return S

To estimate a data value d_u, we use $d_u^* := S[h(u)]$

Theorem: Error Probability of Simple Sketch

Let $\epsilon > 0$ such that $k \geq \frac{2}{\epsilon}$. Then, in the strict turnstile model, the following holds for d_u^*:

1. $d_u \leq d_u^*$
2. $E(d_u^* - d_u) \leq \frac{\epsilon}{2}\|d\|_1$ and $Pr(d_u^* - d_u \geq \epsilon\|d\|_1) \leq \frac{1}{2}$

To reduce the error probability, we create multiple simple sketches and take for each value the smallest estimate resulting from the obtained sketches:

To estimate a data value d_u, we use $d_u* := \min_{j \in [\ell]} S[h_j(u), j]$

Theorem: Error Probability of Count Min Sketch

Let $\epsilon, \delta > 0$ such that $k \geq \frac{2}{\epsilon}$ and $\ell \geq \log \frac{1}{\delta}$. Then, in the strict turnstrile model, the following holds for the estimator d_u^*

1. $d_u \leq d_u^*$
2. $Pr(d_u^* - d_u \geq \epsilon\|d\|_1) \leq \delta$

8.5.1 Heavy Hitters

Definition: Heavy Hitter

An element $u \in \mathbb{U}$ is a heavy hitter with threshold $\tau > 0$ for a data vector d if

$$d_u \geq \tau \|d\|_1$$

Theorem: Error of CM Heavy Hitters

We assume the cash register model. Let $\epsilon, \delta, \tau > 0$ and $k \geq \frac{2}{\epsilon}$ and $\ell \geq \log \frac{n}{\delta}$. Then, the algorithm CM Heavy Hitters(k, ℓ, τ) returns

1. All elements u such that $d_u \geq \tau \|d\|_1$
2. With probability at least $1 - \delta$ no elements u such that $d_u \leq (\tau - \epsilon)\|d\|_1$

Algorithm: CM Heavy Hitters(k, ℓ, τ)

Let $\mathcal{H}$ be a universal family of hash functions from $\mathbb{U}$ to $[k]$.

- Compute a CM sketch S with parameters k, ℓ
- Maintain $\|d\|_1$ during the computation
- During the computation, maintain a set H of elements $u \in \mathbb{U}$ whose estimated value d_u^* is at least $\tau \|d\|_1$
- After each update, remove elements from H whose value has dropped below $\tau \|d\|_1$
- return H