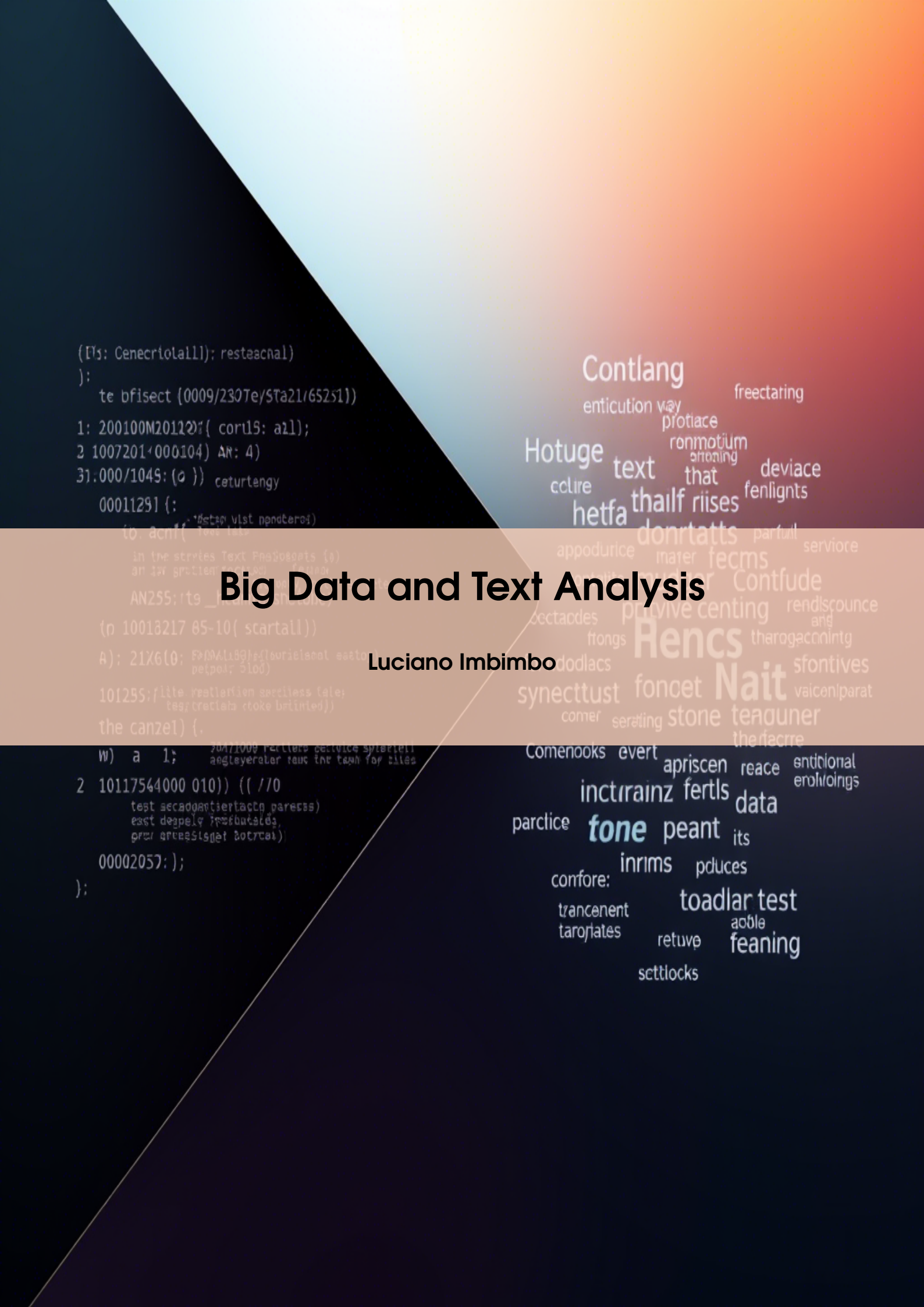


Big Data and Text Analysis

Luciano Imbimbo

PUBLISHED BY RAFAEL BRITO GOMES

BOOK-WEBSITE.COM

Primeira impressão?, Ainda não

Contents

Data Science

1. Data Science

1.1 Data, data everywhere

Data is a torrent flowing into every area of the global economy. Companies churn out a burgeoning volume of transactional data, capturing information about their customers, suppliers, and operations. millions of networked sensors are being embedded in the physical world in devices such as mobile phones, smart energy meters, automobiles, and industrial machines that sense, create, and communicate data in the age of the IoT. Social media sites, smartphones, and other consumer devices including PCs and laptops have allowed billions of individuals around the world to contribute to the amount of big data available.

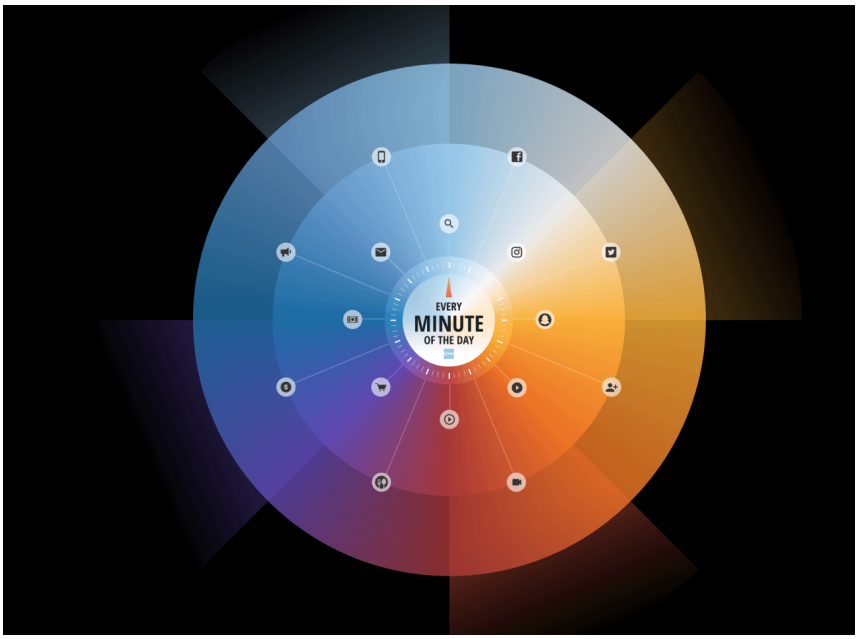


Figure 1.1: Data never sleeps

This huge amount of data led to creation of **Big data**. Big data refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze.
What Does Big Data Look Like?
The three Vs of volume, velocity and variety are commonly used to characterize different aspects of big data.

1.1.1 Volume

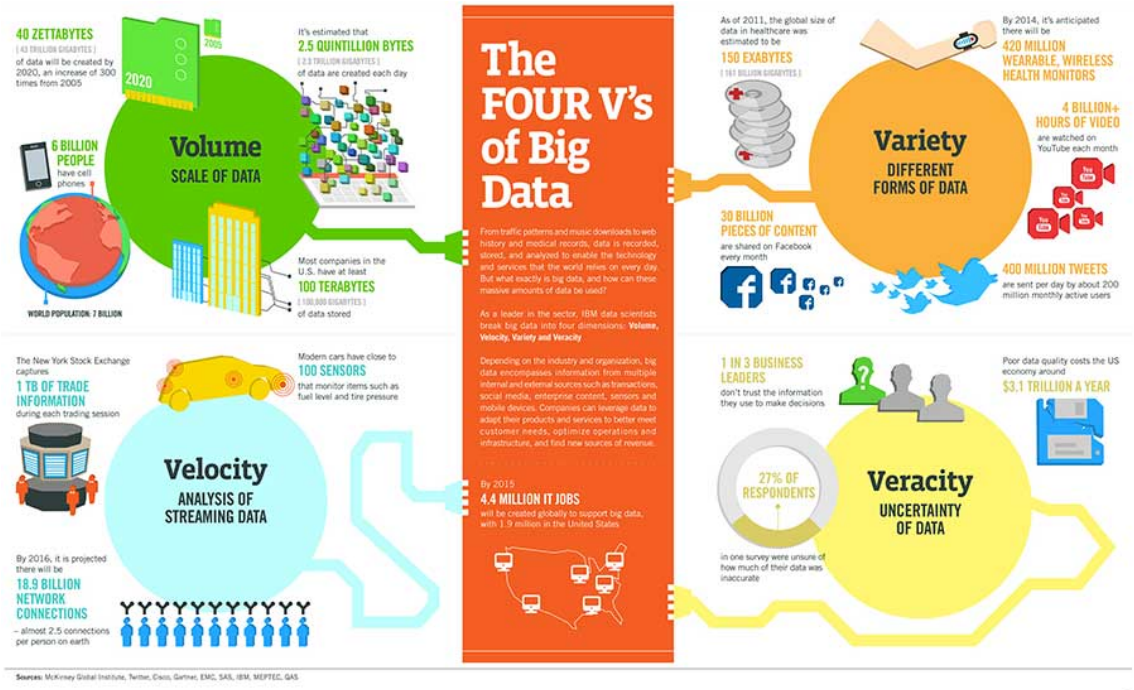
The benefit gained from the ability to process large amounts of information is the main attraction of big data analytics. For this reason the volume presents a challenge to conventional IT structures (scalable storage, and a distributed approach to querying data).

1.1.2 Velocity

The importance of the increasing rate at which data flows into an organization has followed a similar pattern to that of volume. The way we deliver and consume products and services is increasingly instrumented, generating a data flow back to the provider. We have to stream fast-moving data into bulk storage for later batch processing.

1.1.3 Variety

Rarely data is in a form perfectly ordered and ready for processing; a common theme in big data systems is that the source data is diverse, and doesn't fall into neat relational structures. So a common use of big data processing is to take unstructured data and extract ordered meaning, for consumption either by humans or as a structured input to an application



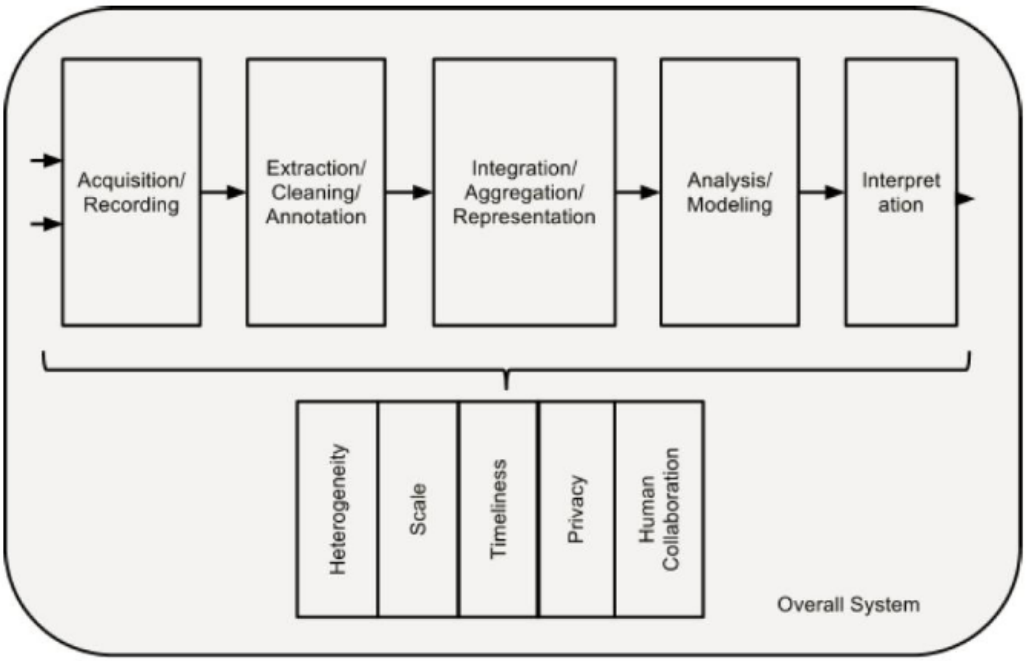
Thanks to Big Data companies can create different values:

- Creating transparency, making big data easily accessible to relevant stakeholders.
- Enabling experimentation to discover needs, and improve performance.
- Segmenting populations to customize actions.

- Replacing/supporting human decision making with automated algorithm and unearth valuable insights that would otherwise remain hidden.
- Innovating new business models, products, and services

1.2 The Big Data analysis pipeline

Major steps in the analysis of Big Data are in the top half of the figure below. Instead, the bottom half of the figure shows Big Data characteristics that make these steps challenging.



1.2.1 Data acquisition

Data acquisition has been understood as the process of gathering, filtering, and cleaning data before the data is put in a data warehouse or any other storage solution.

1.2.2 Information extraction and cleaning

Frequently, the information collected is not in a format ready for analysis: pull out the required information from the underlying sources and express it in a structured form suitable for analysis.

1.2.3 Data integration, aggregation, and representation

Effective large-scale analysis often requires the collection of heterogeneous data from multiple sources. Data transformation and integration tools help the data analyst to resolve heterogeneities in data structure and semantics.

1.2.4 Modeling and analysis

Methods for querying and mining Big Data are fundamentally different from traditional statistical analysis on small samples because Big Data is often noisy, dynamic, heterogeneous, inter-related, and untrustworthy. Nevertheless, with suitable statistical care, one can use approximate analyses to get good results without being overwhelmed by the volume.

1.2.5 Interpretation

A decision-maker has to interpret the results of an analysis. This involves examining all the assumptions made and retracing the analysis (check possible sources of error). A system must provide users with the ability to interpret analytical results and to repeat the analysis with different assumptions, parameters, or datasets to support the human thought.

1.3 Dealing with Big Data

We now turn to some common challenges that underlie many, and sometimes all, of the phases of Big Data pipeline.

1.3.1 Heterogeneity

When humans consume information, a great deal of heterogeneity is tolerated, instead a machine analysis algorithms expect homogeneous data, and cannot understanding nuances. In consequence, data must be carefully structured as a first step in data analysis. An associated challenge is to automatically generate the right metadata to describe the data recorded. Recording information about the data at its birth is not useful unless this information can be interpreted and carried along through the data analysis pipeline. This is called **data provenance**.

1.3.2 Inconsistency and incompleteness

Big Data includes information provided by diverse sources, of varying reliability and so uncertainty, errors, and missing values are endemic, and must be managed. On the bright side, the volume and redundancy of Big Data can be exploited to compensate for missing data, to crosscheck conflicting cases, to validate trustworthy relationships and to uncover hidden relationships and models.
Even after error correction has been applied, some incompleteness and some errors in data are likely to remain that must be managed during data analysis.

1.3.3 Data Quality

Data is frequently missing or incongruous. If data is incongruous, do you decide that something is wrong with badly behaved data, or that the incongruous data is telling its own story, which may be more interesting? It's usually impossible to get "better" data, and you have no alternative that work with the data at hand.

1.3.4 Scale

Managing large and rapidly increasing volumes of data has been a challenging issue for many decades. Due to power constraints, clock speeds have largely stalled and processors are being built with increasing numbers of cores enabling parallelism.
Traditional relational database systems stop being effective at this scale and so most of the organizations have found it necessary to go beyond the relational database model. NoSQL databases, or Non-Relational databases are now used to manage huge amounts of data. Obviously, storing data is only part of building a data platform because data is only useful if you can do something with it. Enormous datasets present computational problems to resolve (MapReduce).

1.3.5 Timeliness

The fundamental challenge is to provide interactive response times to complex queries at scale over high-volume event streams. So the challenge is to find elements in a very large dataset that meet a specified criterion and do it quickly.

1.3.6 Privacy and data ownership

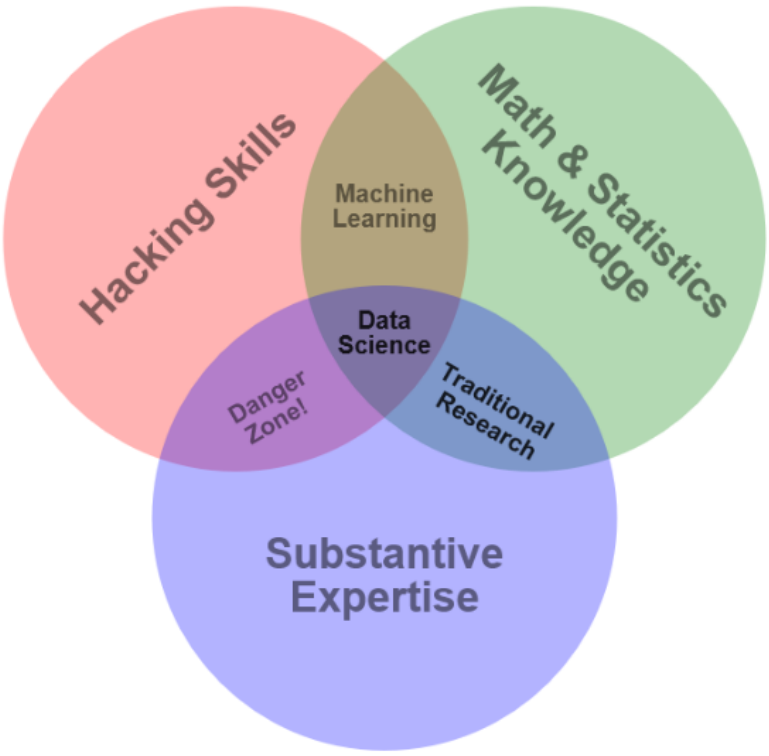
There is great public fear regarding the inappropriate use of personal data, particularly through linking of data from multiple sources. The way the data owned by an organization are stored can be a central strategic consideration.

1.3.7 The human perspective: Visualization and collaboration

To fully reach potential of Big Data, we need to consider scale not just for the system but also from the perspective of humans. We have to make sure the end points (humans) can properly "absorb" the results of the analysis and not get lost in a sea of data. For these reasons Big Data Analytics has to be designed to have a human in the loop and enable visual analytics to have a better comprehension of results.

1.4 What is data science?

Data science is the study of data to extract meaningful insights for business. It is a multidisciplinary approach that combines principles and practices from the fields of mathematics, statistics, artificial intelligence, and computer engineering to analyze large amounts of data.



Survey published on 2017 describing Data Science employees and enthusiasts working at Microsoft.
Four aspects have been analyzed:

- Who are Data Scientists at Microsoft?
  - Professional experience (13.6 years of professional experience on average).
  - Job title (software engineers, program managers, data science discipline).
  - Education (master's degree, PhDs, bachelor's degree).
  - Skill (Business, ML/Big Data, Math and Operations Research, Programming, and Statistics).
  - Working styles

- What Problem Topics do Data Scientists Work on?
  - User engagement: with usage data from software, teams pay attention to how customers use their software like which features are most often used.
  - Software productivity and quality: study performance modelling, requirements elicitation, bug finding and repair, release planning, and system health and service quality monitoring.
  - Domain-specific problems: assessing a speech-based natural language processing platform; creating predictive models for stock pricing; modeling advertiser value for ads; identifying malware; predicting power consumption and so on.
  - Business intelligence.
- What challenges do data scientists face?
  - Data Quality
  - Data Availability
  - Data Preparation
  - Scale
  - Machine Learning
- How do Data Scientists ensure the quality of their work?
  - **Cross Validation**: developers use multiple data sources to triangulate their results and perform a held-out comparison with respect to different sources of data.
  - Check Data Distribution: to build intuition about data, data scientists explore and understand the underlying distributions by computing descriptive statistics and measuring confidence intervals.
  - Dogfood Simulation: data scientists can create new data through simulation.
  - Type and Schema Checking: to ensure data quality and integrity, data scientists often check their format, type and schema to see whether individual fields are well-defined.
  - Repeatability: to increase confidence in the correctness of results when processing and ingesting data, data scientists repeat the same procedure multiple times to replicate the same results.
  - Check Implicit Constraints.

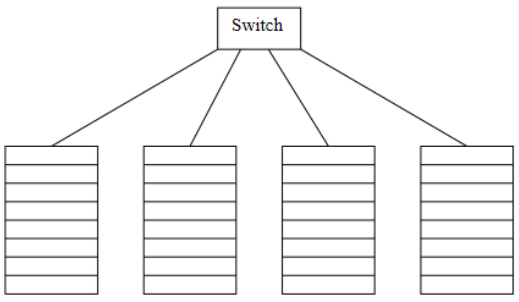
2. Map Reduce

Modern data-mining applications, often called “big-data” analysis, require us to manage immense amounts of data quickly.To deal with applications such as these, a new software stack has evolved:

1. **Distributed file system**
2. **Map-Reduce**
3. **New platforms** (Spark, Flink)

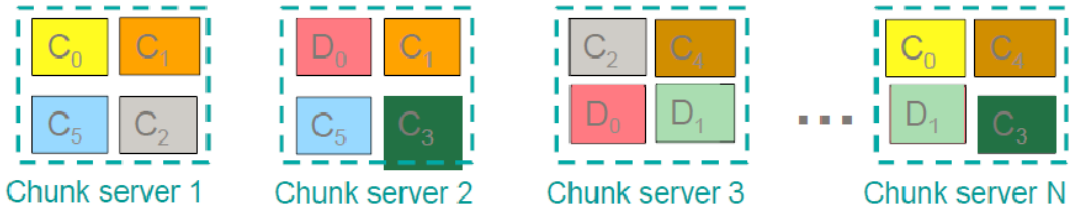
2.1 Distributed File Systems

In the past, applications that called for parallel processing, such as large scientific calculations, were done on special-purpose parallel computers with many processors and specialized hardware. However, the prevalence of large-scale Web services has caused more and more computing to be done on installations with thousands of compute nodes operating more or less independently. In these installations, the compute nodes are commodity hardware, which greatly reduces the cost compared with special-purpose parallel machines.
The new parallel-computing architecture, sometimes called cluster computing, is organized as follows. Compute nodes are stored on racks, perhaps 8–64 on a rack. The nodes on a single rack are connected by a network, typically gigabit Ethernet. There can be many racks of compute nodes, and racks are connected by another level of network or a switch.



To exploit cluster computing, files must look and behave somewhat differently from the conventional file systems found on single computers. This new file system, often called a distributed file system or DFS is typically used as follows.

- Files are divided into **chunks**; chunks are replicated at different compute nodes. Moreover, the nodes holding copies of one chunk should be located on different racks, so we don't lose all copies due to a rack failure. Normally, both the chunk size and the degree of replication can be decided by the user
- Another small file, called **Master node** or namenode, maintains the filesystemtree, metadata and directories.



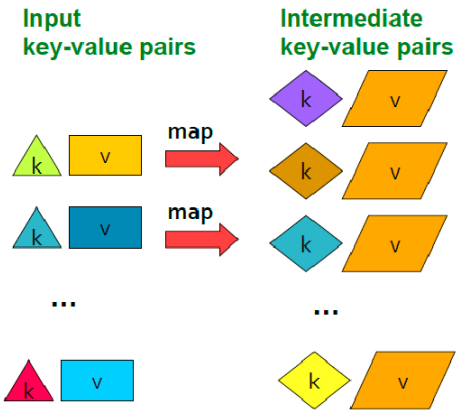
2.2 Map-Reduce

MapReduce is a style of computing that has been implemented in several sys- tems, including Google's internal implementation (simply called MapReduce) and the popular open-source implementation Hadoop.

1. Some number of **Map** tasks each are given one or more chunks from a distributed file system. These Map tasks turn the chunk into a sequence of key-value pairs.
2. The key-value pairs from each Map task are collected by a master controller and **sorted by key**. The keys are divided among all the Reduce tasks, so all key-value pairs with the same key wind up at the same Reduce task.
3. The **Reduce** tasks work on one key at a time, and combine all the values associated with that key in some way.

2.2.1 The MAP step

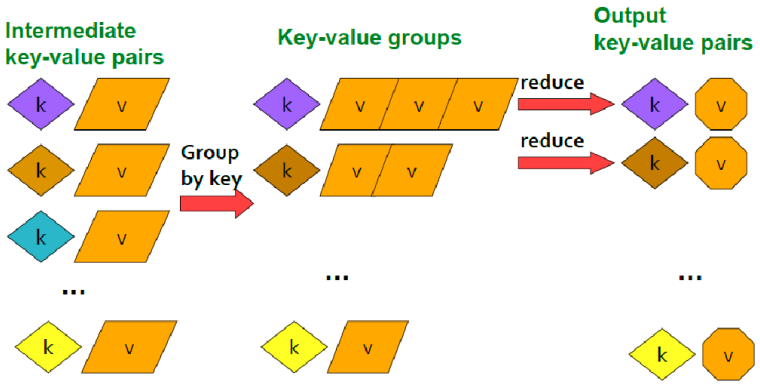
The Map function takes an input element, a tuple or a document, as its argument and produces zero or more key-value pairs (keys are not "keys" in the usual sense, they do not have to be unique).



As soon as the Map tasks have all completed successfully, the key-value pairs are grouped by key, and the values associated with each key are formed into a single list of values for that key.

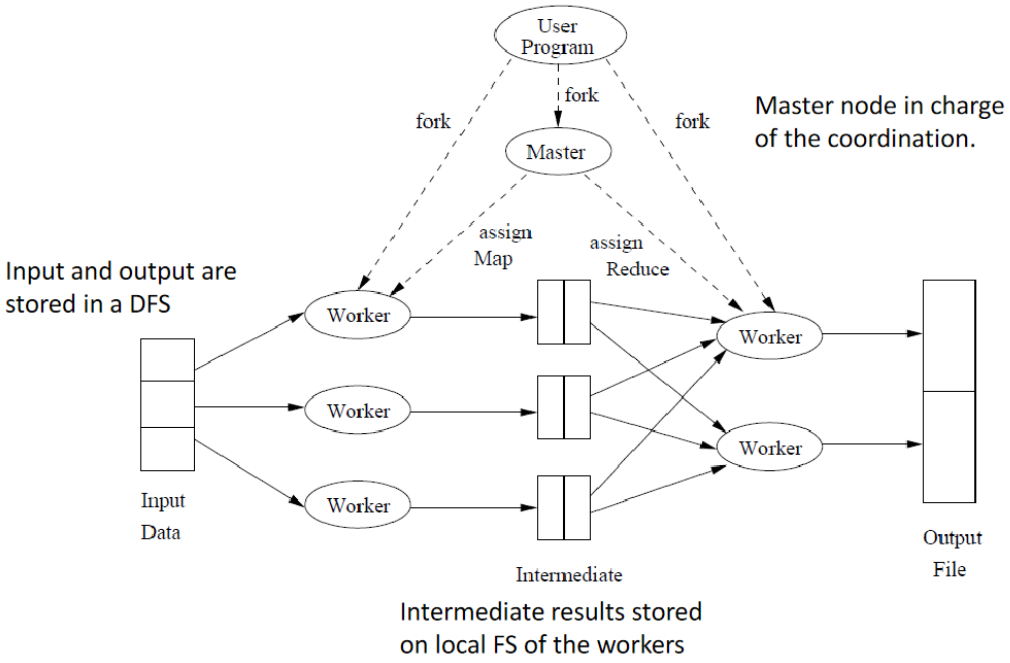
2.2.2 The REDUCE step

The Reduce function's argument is a pair consisting of a key and its list of associated values. The output of the Reduce function is a sequence of zero or more key-value pairs. These key-value pairs can be of a type different from those sent from Map tasks to Reduce tasks, but often they are the same type.



2.2.3 Details of Map-Reduce Execution

Input and output of map-reduce process are both key-value pairs because, in this way, we have the possibility of creating a chain of map-reduce.

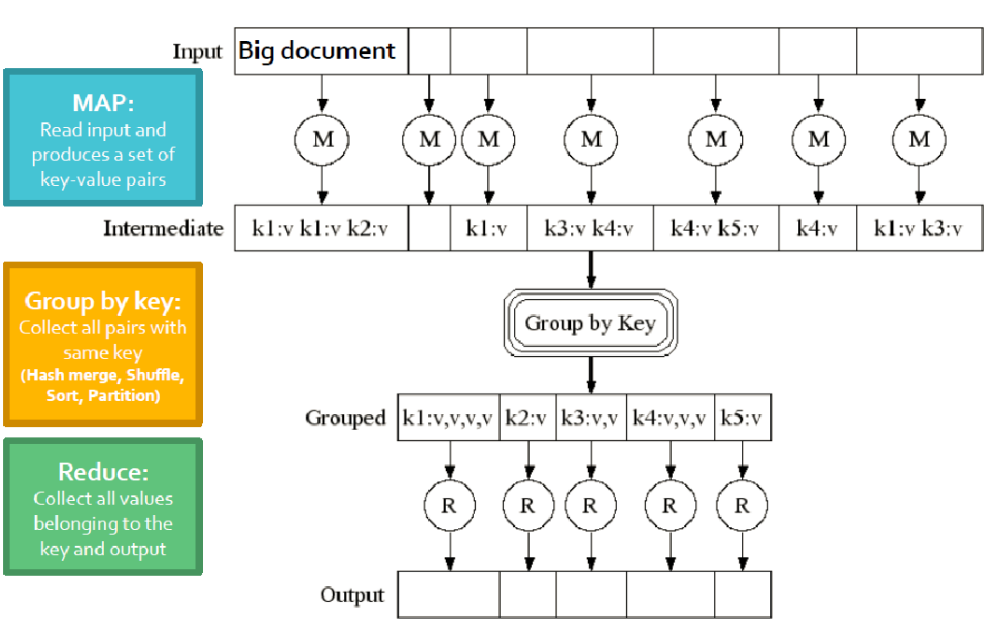


Intermediate results are stored on the local file system of the workers, and only the master node is aware of them. It is responsible for coordination. For this reason, the higher cost of map-reduce is

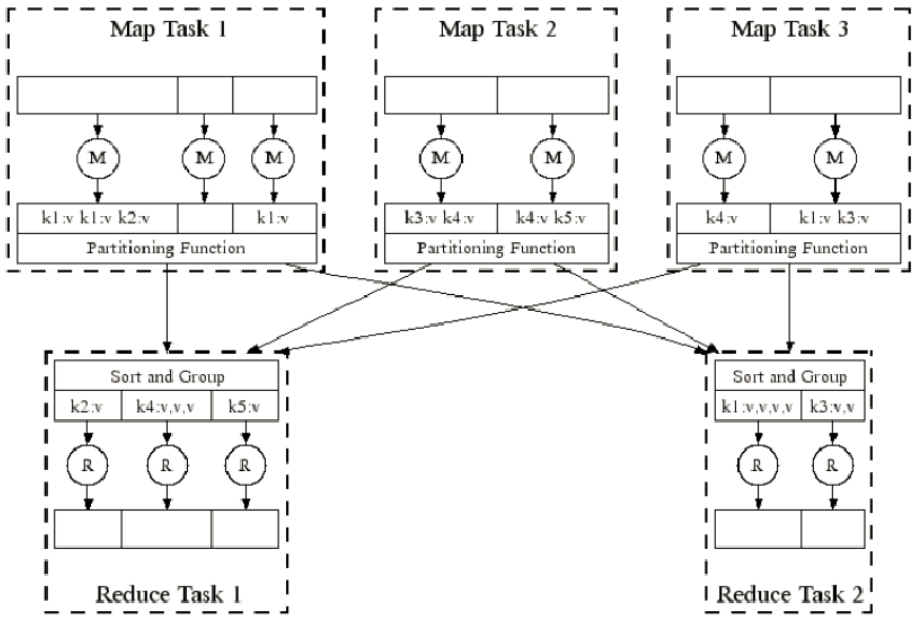
related on the transport of data between the workers; the master-node apply some procedures to optimize traffic and amount of data transported.
In case of failure of a map/reduce worker there is no problem because the tast is simply restarted. Instead, in case of master failure, MapReduce task is aborted and client is notified; for this reason the master node is itself replicated.

2.2.4 Example - Word Count

We have a huge text document and we want to count the number of times of each distinct word appears in the file. We can do it with map reduce as shown in the figure below.



Obviously we can do it with a certain level of parallelism to enhnace general performance.



For achieving maximum parallelism we could use one Reduce task to execute each reducer but this

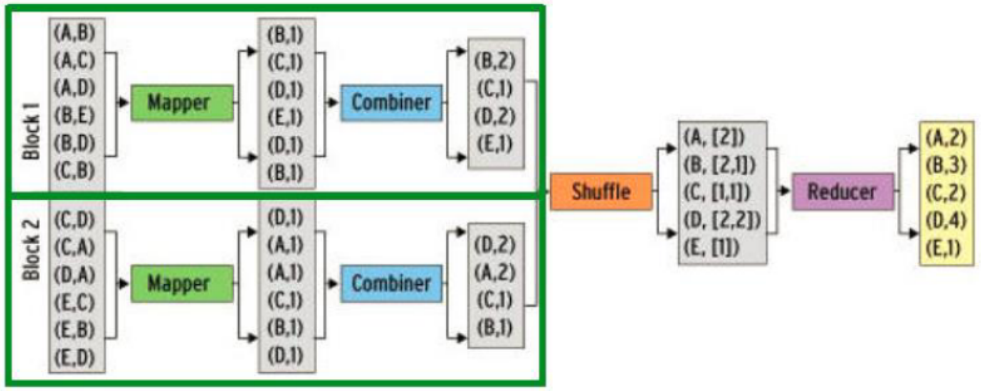
plan is not usually the best because there is overhead associated with each task we create and there could be far more keys than there are compute nodes available. To balance this trade-off we can use the following measures:

- The **reducer size (q)** is the upper bound on the number of values that are allowed to appear in the list associated with a single key (with combiner is always 1).
- The **replication rate (r)** is the number of key-value pairs produced by all the Map tasks on all the inputs, divided by the number of inputs.

The tradeoff between q and r is as follows: when q is a large number, r will be small, and when r is a large number, q will be small.
Other applications of map-reduce can be find in my "Algorithms Map-Reduce" document.

2.2.5 Combiners

In case the reduce function is associative and commutative (count), values can be combined in any order with the same result. In these case, to reduce data transported between workers, we can use a **combiner** that apply reduce task during the map task.



3. Data Mining

In today's digital era, the ability to analyze and derive insights from large datasets is crucial. Data mining and machine learning are at the forefront of this revolution, enabling businesses and researchers to make sense of vast amounts of data. This document explores the fundamental concepts, methodologies, and applications of data mining and machine learning, focusing on their role in Big Data Analytics. First of all, what are the difference between Machine Learning and Data Mining?

3.0.1 Machine Learning

Machine learning (ML) is the study of algorithms that improve automatically through experience. It involves the construction of systems that can learn from data and make decisions with minimal human intervention. The key components of machine learning systems include:

- **Learning Algorithms**: These are the core of ML systems and are used to create models that can make predictions or decisions without being explicitly programmed.
- **Training Data**: This is the data used to train the learning algorithms. It must be representative of the problem domain to ensure the model generalizes well to new data.
- **Model Evaluation**: This involves assessing the performance of the learning model using various metrics and testing it on unseen data.

Machine learning is applied in numerous domains such as web search, spam filters, recommender systems, ad placement, credit scoring, fraud detection, and stock trading.

3.0.2 Data Mining

Instead, Data mining, also known as Knowledge Discovery in Databases (KDD), involves the process of discovering patterns and knowledge from large amounts of data. It is an interdisciplinary field that combines techniques from statistics, machine learning, and database management. Key steps in data mining include:

1. **Data Cleaning**: Removing noise and inconsistencies from the data.
2. **Data Integration**: Combining data from multiple sources.
3. **Data Selection**: Selecting relevant data for the analysis.

4. **Data Transformation**: Transforming data into suitable formats for mining.
5. **Data Mining**: Applying algorithms to extract patterns from the data.
6. **Pattern Evaluation**: Identifying the truly interesting patterns representing knowledge.
7. **Knowledge Presentation**: Using visualization and knowledge representation techniques to present the mined knowledge to users.

In conclusion we can say that ML use Data mining operations to extract information/knowledge from the data but, most of time, these 2 words are synonym.

Statistical Limits on Data Mining

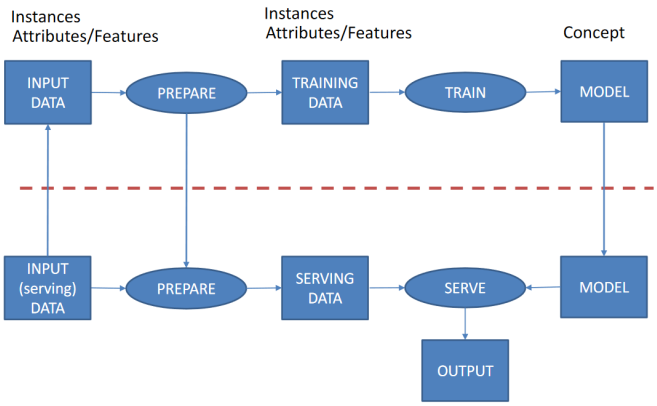
A crucial aspect of data mining is distinguishing between genuine patterns and random occurrences. Bonferroni's Principle provides a way to avoid false positives by adjusting the significance level when multiple comparisons are made. This principle is essential in ensuring the robustness of the findings from data mining processes.

Suppose there are believed to be some "evil-doers" out there, and we want to detect them. Suppose further that we have reason to believe that periodically, evil-doers gather at a hotel to plot their evil. There are one billion people who might be evil-doers and everyone goes to a hotel one day in 100. A hotel holds 100 people; hence, there are 100,000 hotels enough to hold the 1% of a billion people who visit a hotel on any given day. We shall examine hotel records for 1000 days. To find evil-doers in this data, we shall look for people who, on two different days, were both at the same hotel. The result with this data is 250.000 people who like evil-doers and obviously not all these pairs are true (false positive can be random).

3.1 The Machine Learning Pipeline

The machine learning pipeline consists of several stages, each crucial for developing a robust ML model:

1. **Input Data Preparation**: This involves collecting raw data, cleaning it, and transforming it into a format suitable for analysis. Key tasks include handling missing values, normalizing features, and encoding categorical variables.
2. **Training Data Preparation**: Splitting the data into training and testing sets.
3. **Model Training**: Using learning algorithms to train models on the training data. This step involves selecting the appropriate algorithm and tuning its hyperparameters.
4. **Model Evaluation**: Assessing the model's performance using metrics such as accuracy, precision, recall, and F1 score.
5. **Model Deployment**: Integrating the trained model into a production environment where it can make predictions on new data.



3.1.1 Concepts in Machine Learning

Machine learning concepts are categorized into different types of learning:

- **Classification Learning**: This involves predicting a discrete class label. Examples include email spam detection and image recognition.
- **Numeric Prediction (Regression)**: This involves predicting a continuous value. Examples include predicting house prices and stock market trends.
- **Clustering**: This involves grouping similar instances into clusters. It is often used in customer segmentation and image compression and it required an explanation on how it creates the clusters (on which features).
- **Data Reduction**: This involves reducing the size of the data while preserving important information.
- **Association Learning**: This involves finding associations between features. It is used in market basket analysis to identify products frequently bought together.
- **Similarity Matching**: This involves identifying similar individuals based on data. It is used in recommendation systems and fraud detection.
- **Profiling**: This involves characterizing the typical behavior of an individual, group, or population. It is used in behavior analysis and anomaly detection.
- **Link Prediction**: This involves predicting connections between data items. It is used in social network analysis and recommendation systems.
- **Causal Modeling**: This involves understanding what events or actions actually influence others. It is used in epidemiology and econometrics.

Supervised vs. Unsupervised Methods

Supervised learning involves training a model on labeled data, where the outcome or target variable is known. Techniques include:

- **Classification** is a supervised learning task where the goal is to predict the class label of a given instance. Common algorithms used for classification include:
  - **Decision Trees**: These create a tree-like model of decisions and their possible consequences. They are easy to interpret and can handle both numerical and categorical data.
  - **Support Vector Machines (SVMs)**: These find the hyperplane that best separates the data into different classes. They are effective for high-dimensional spaces.
  - **Naive Bayes**: This is based on Bayes' theorem and assumes independence between predictors. It is simple and effective for many tasks.
- **Regression** is a type of supervised learning where the goal is to predict a continuous outcome. Common algorithms used for regression include:
  - **Linear Regression**: This assumes a linear relationship between the input features and the target variable.
  - **Polynomial Regression**: This models the relationship between the input features and the target variable as a polynomial.
  - **Ridge and Lasso Regression**: These are extensions of linear regression that include regularization to prevent overfitting.

Instead, unsupervised learning involves training a model on unlabeled data, aiming to find hidden patterns. Techniques include:

- **Clustering** is an unsupervised learning task where the goal is to group similar instances into clusters. Common algorithms used for clustering include:
  - **K-means Clustering**: This partitions the data into k clusters, where each data point belongs to the cluster with the nearest mean.
  - **Hierarchical Clustering**: This builds a hierarchy of clusters using either a bottom-up

or top-down approach.
  - **DBSCAN**: This is a density-based clustering algorithm that can find arbitrarily shaped clusters.
- **Association learning** aims to discover interesting relationships between variables in large datasets. The most common example is market basket analysis, which identifies items frequently bought together. Algorithms used for association learning include:
  - **Apriori Algorithm**: This identifies frequent itemsets and generates association rules.
  - **FP-Growth Algorithm**: This is an efficient alternative to Apriori that uses a tree structure to find frequent itemsets.

3.2 Instances and Attributes

Instances, also known as data points or records, are the fundamental units of data in a dataset. Each instance is characterized by attributes or features, which can be:

- **Numerical**: Continuous values like age, salary, temperature.
- **Categorical**: Discrete values like gender, color, type.
- **Ordinal**: Categorical values with an order like rankings or ratings.

A common issue is flattening data, also known as denormalization, that involves converting relational data into a flat file format. This process simplifies data analysis but can introduce spurious correlations that reflect the database's structure rather than meaningful relationships.

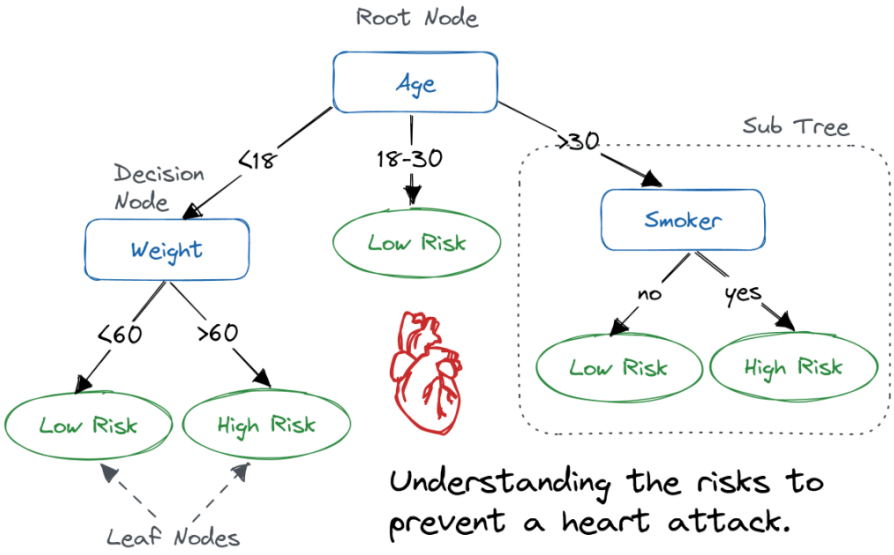
Name	Gender	Parent1	parent2
Peter	Male	?	?
Peggy	Female	?	?
Steven	Male	Peter	Peggy
Graham	Male	Peter	Peggy
Pam	Female	Peter	Peggy
Ian	Male	Grace	Ray
Pippa	Female	Grace	Ray
Brian	Male	Grace	Ray
Anna	Female	Pam	Ian
Nikki	Female	Pam	Ian

Figure 3.1: A right representation of input data can drive performance of the model

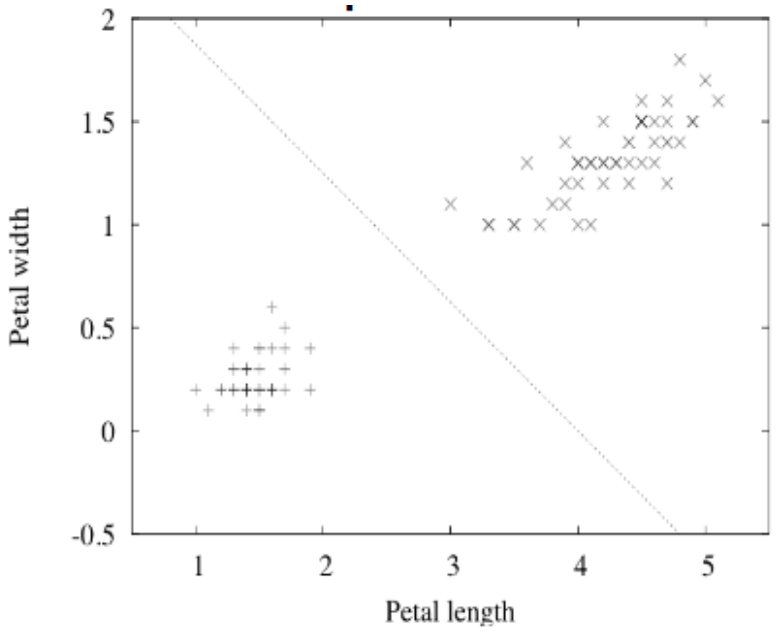
3.3 Building Machine Learning Models

Various algorithms can be used to build machine learning models. The choice of algorithm depends on the problem and the nature of the data. Common algorithms include:

- **Decision Trees**: Decision trees use a divide-and-conquer approach to classify data points. They split the data based on attribute values, creating branches that lead to a decision or classification. Decision trees are easy to interpret and can handle both numerical and categorical data. However, they can be prone to overfitting, which can be mitigated by techniques such as pruning.



- **Classification rules**: a popular alternative to decision trees, consists of a series of tests, just like the tests at the nodes of a decision tree, that verify general logical expressions. It is very easy converting a tree into a set of rules because one rule correspond to a leaf.
- **Linear Models**: Linear models, such as linear regression, predict outcomes based on a linear combination of input features. Linear models assume a linear relationship between the input features and the target variable and separates the two classes with a line, called **decision boundary**, that becomes a high-dimensional plane (hyperplane) when there are multiple attributes.



- **Instance-based**: Instance-based learning algorithms, like k-nearest neighbors (k-NN), classify data points based on their similarity to other instances in the dataset. These methods are simple and effective but can be computationally intensive for large datasets. They rely on distance metrics, such as Euclidean distance, to measure similarity between instances.

Optimization and Overfitting

A significant challenge in machine learning is finding the right balance between fitting the training data well and generalizing to new data. Overfitting occurs when a model learns the noise in the training data rather than the underlying pattern. Techniques like cross-validation, regularization, and pruning help mitigate overfitting.

3.3.1 One Rule

The One-Rule (1R) classifier is a simple yet surprisingly effective classification algorithm. It constructs a one-level decision tree, which is equivalent to a single rule for classification.

1. **Attribute Evaluation**: for each attribute, generate a rule that classifies instances based on the values of that attribute. For each value of the attribute, determine the most frequent class (mode) among the training instances that have that value.
2. **Error Calculation**: calculate the error rate of the rule for each attribute. The error rate is the proportion of instances that do not belong to the majority class for their corresponding attribute value.
3. **Rule Selection**: select the attribute with the lowest error rate. This attribute and its associated rule become the One-Rule classifier.

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Attribute	Rules	Errors	Total errors
Outlook	Sunny → No	2/5	4/14
	Overcast → Yes	0/4	
	Rainy → Yes	2/5	
Temp	Hot → No*	2/4	5/14
	Mild → Yes	2/6	
	Cool → Yes	1/4	
Humidity	High → No	3/7	4/14
	Normal → Yes	1/7	
Windy	False → Yes	2/8	5/14
	True → No*	3/6	

* Random choice between two equally likely outcomes

The algorithm is straightforward to understand and implement and perform not much worse than much more complex decision trees. Problems of 1R are related to numeric attributes, where, given the fluctuations in values, we are unable to extract a general rule (very sensitive to noise and prone to overfitting). We have two main solutions for this problem:

- Enforce minimum number of instances in majority class per interval.
- Build one rule for each class as a conjunction of tests, one for each attribute and create a system of weights.

3.3.2 Naive Bayes

Statistical modeling is a fundamental aspect of data mining and machine learning. It involves creating models that use statistical methods to predict outcomes based on input data. Two assumptions

are made to apply these classifiers:

1. Statistical models often assume that all attributes (features) in the data are equally important. This means each feature contributes independently to the outcome without being influenced by other features.
2. Another assumption is that attributes are statistically independent given the class value. This means that knowing the value of one attribute does not give any information about the value of another attribute when the class is known. The independence assumption, although rarely correct in real-world scenarios, simplifies the modeling process. It allows the use of algorithms that can handle large datasets efficiently while still providing reasonable performance.

The Naive Bayes classifier is a prime example of a statistical model that relies on the independence assumption. It calculates the probability of a class based on the product of the individual probabilities of the features.

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)} = \frac{P(E_1|H)P(E_2|H)\ldots P(E_n|H) \cdot P(H)}{P(E)} \tag{3.1}$$

where:

- $P(H|E)$ is the posterior probability of hypothesis H given the evidence E.
- $P(E|H)$ is the likelihood of evidence E given that hypothesis H is true.
- $P(H)$ is the prior probability of hypothesis H.
- $P(E)$ is the prior probability of evidence E.

Outlook			Temperature			Humidity			Windy			Play	
	Yes	*No*		*Yes*	*No*		*Yes*	*No*		*Yes*	*No*	*Yes*	*No*
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/	5/
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5	14	14
Rainy	3/9	2/5	Cool	3/9	1/5								

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

← ***Evidence E***

$$Pr[yes \mid E] = Pr[Outlook = Sunny \mid yes]$$

Probability of class "yes" ↗

$$\times Pr[Temperature = Cool \mid yes]$$

$$\times Pr[Humidity = High \mid yes]$$

$$\times Pr[Windy = True \mid yes]$$

$$\times \frac{Pr[yes]}{Pr[E]}$$

$$= \frac{\frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14}}{Pr[E]}$$

What if an attribute value doesn't occur with every class value? Probability will be zero and a posteriori probability will also be zero. A remedy can be Laplace estimator, add 1 to the count for every attribute value-class combination so probabilities will never be zero. Statistical models also provide mechanisms for handling missing values. During training, instances with missing values

can be excluded from frequency counts. During classification, attributes with missing values can be omitted from the probability calculations. For numeric attributes, statistical models often assume a normal (Gaussian) distribution. The probability density function for a normal distribution is characterized by its mean and standard deviation. This assumption helps in estimating the likelihood of different outcomes based on the observed data.
Despite its simplicity and the unrealistic independence assumption, the Naive Bayes classifier performs surprisingly well in practice, particularly in text classification problems.

3.4 Decision Tree

Recursive divide-and-conquer fashion.

1. Select attribute for root node and create branch for each possible attribute value
2. Split instances into subsets (one for each branch extending from the node)
3. Repeat recursively for each branch, using only instances that reach the branch

How can we select the best start attribute? A common measure is the purity of attribute. It may be measured through the "information" needed to specify whether a new instance should be classified yes / no, given that the example reached the node. **Entropy** can gives the information required in bits:

$$entropy(p_1, p_2, \ldots, p_n) = -p_1 log(p_1) - p_2 log(p_2) \cdots - p_n log(p_n) \tag{3.2}$$

Outlook = Sunny :

$$\text{info}([2,3]) = entropy(2/5, 3/5) = -2/5 \log(2/5) - 3/5 \log(3/5) = 0.971 bits$$

Outlook = Overcast :

$$\text{info}([4,0]) = entropy(1,0) = -1\log(1) - 0\log(0) = 0 bits$$

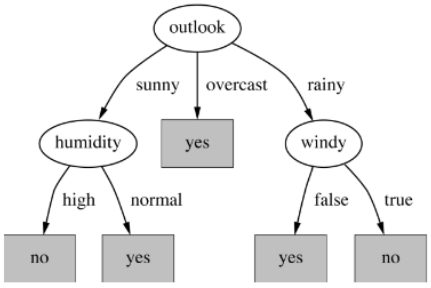
Outlook = Rainy :

$$\text{info}([3,2]) = entropy(3/5, 2/5) = -2/5 \log(2/5) - 3/5 \log(3/5) = 0.971 bits$$

Expected information for attribute:

$$\text{info}([3,2],[4,0],[3,2] = (5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971 = 0.693 bits$$

Then we compute information gain: information before splitting – information after splitting and we decide the best attribute.



A problem can be attributes with a large number of values, for example ID code, because subsets are more likely to be pure if there is a large number of values. To avoid this problem we can use **gain ratio**, a modification of the information gain that reduces its bias. Gain ratio takes number and size of branches into account when choosing an attribute and it corrects the information gain by taking the intrinsic information of a split into account.

3.4.1 Covering algorithms

Convert decision tree into a rule set is, usually, straightforward, but rule set overly complex. Instead, we can generate rule set directly with a covering approach, where at each stage a rule is identified that "covers" some of the instances. Generates a rule by adding tests that maximize rule's accuracy:

- t total number of instances covered by rule
- p positive examples of the class covered by rule
- t – p number of errors made by rule
- select test that maximizes the ratio p/t

We are finished when p/t = 1 or the set of instances can't be split any further.

Rule we seek:

```
If ?
    then recommendation = hard
```

Possible tests:

Age = Young	2/8
Age = Pre-presbyopic	1/8
Age = Presbyopic	1/8
Spectacle prescription = Myope	3/12
Spectacle prescription = Hypermetrope	1/12
Astigmatism = no	0/12
Astigmatism = yes	4/12
Tear production rate = Reduced	0/12
Tear production rate = Normal	4/12

3.4.2 Mathematical functions

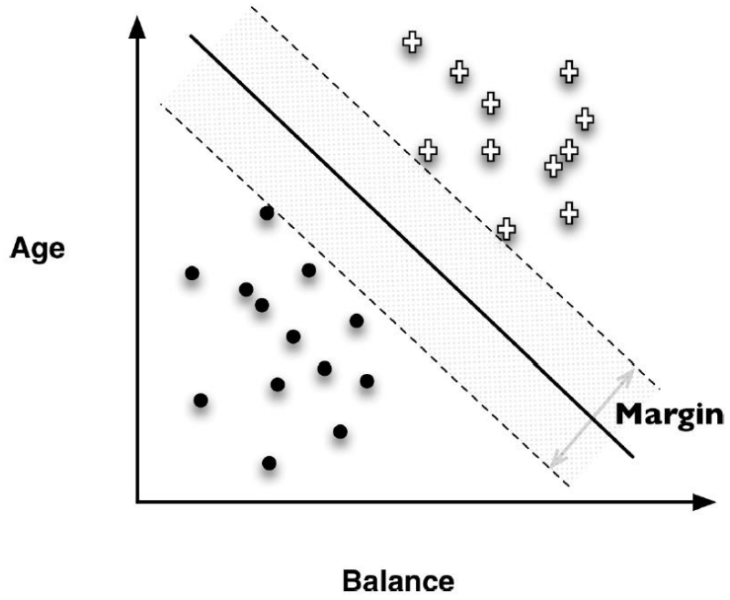
Many different possible linear boundaries can separate the two groups of points. The simplest approach is to use a linear discriminant, a function of the decision boundary that is a linear combination, a weighted sum, of the attributes.

$$x = w_0 + w_1 a_1 + w_2 a_2 + \ldots w_k a_k \tag{3.3}$$

Linear regression, logistic regression, and support vector machines are all very similar instances of linear models to data. The key difference is that each uses a different objective function.

Support Vector Machines (SVMs)

The key idea is to maximize the margin around the separating hyperplane, which helps in achieving better generalization.



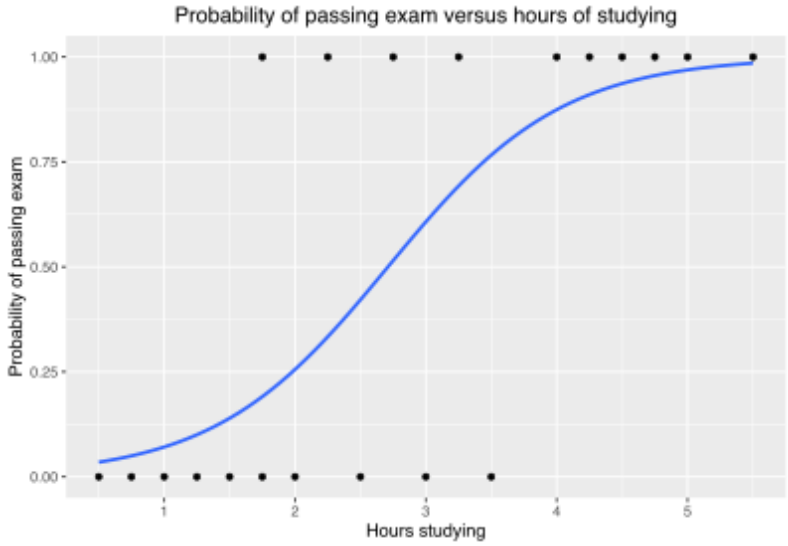
Logistic Regression

The key idea behind logistic regression is to model the log-odds of the probability of an event occurring as a linear combination of one or more independent variables. The log-odds (also known as the logit) is the natural logarithm of the odds, where the odds are the ratio of the probability of the event occurring to the probability of the event not occurring.

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k \tag{3.4}$$

where:

- p is the probability of the event occurring.



Linear Regression

The linear regression model structure is the same as for the linear discriminant function. Therefore, an intuitive notion of the fit of the model is: how far away are the estimated values from the true values on the training data? We would like to minimize this error.

- **Absolute error**: minimize the distance in absolute value (input and output same scale).
- **Squared error**: minimize the distance squared (always positive values and small values worth less)

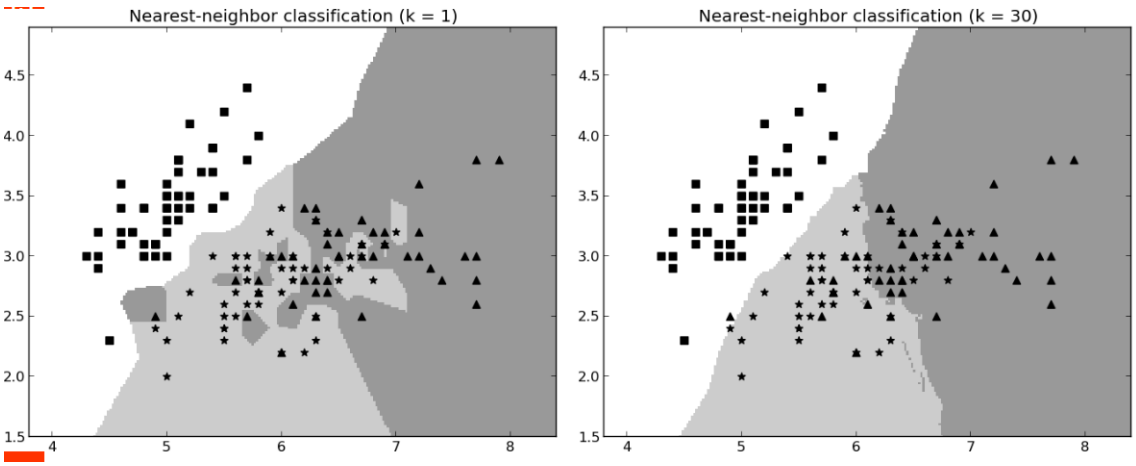
3.4.3 Instance-based learning

It classifies data points based on their similarity to other instances in the dataset. These methods are simple and effective but can be computationally intensive for large datasets. They rely on distance metrics, such as Euclidean distance or city-block metric, to measure similarity between instances. Usually, different attributes are measured on different scales so need to be normalize to don't have values much bigger than others. Instead, usually policy for missing values, is that are assumed to be maximally distant.

The basic procedure is simple, given a new example whose target variable we want to predict, we scan through all the training examples and choose several that are the most similar to the new example. Then we predict the new example's target value, based on the nearest neighbors' (known) target value and using some combining function (like voting or averaging), operating on the neighbors' known target values, that give us a prediction. How many neighbors should be used?

- With small values of K, we will have better performance but greater tendency to overfit
- If we increase k to the maximum possible (so that k = n) the entire dataset would be used for every prediction (less overfitting but worse performance)

Should we treat all neighbors the same? Nearest-neighbor methods often use weighted voting or similarity moderated voting such that each neighbor's contribution is scaled by its similarity.



3.4.4 Clustering

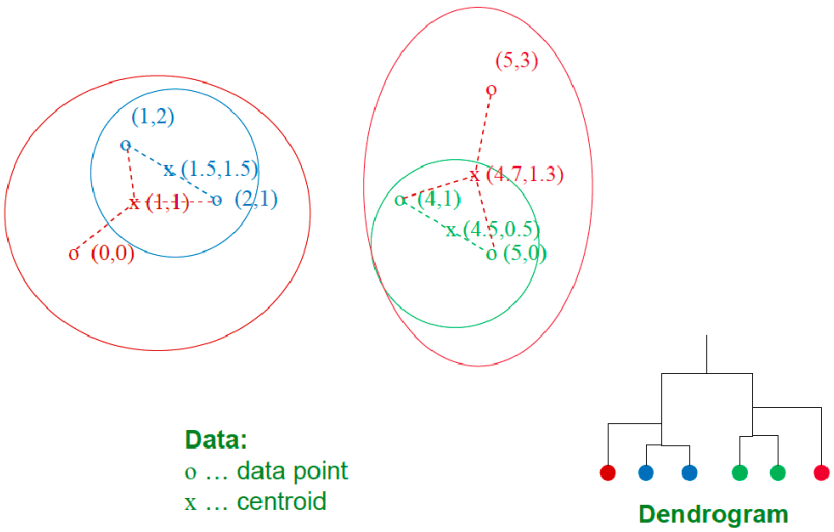
Clustering techniques apply when there is no class to be predicted and aim to divide instances into "natural" groups. We can divide clustering algorithms into two groups that follow two fundamentally different strategies.

- **Hierarchical or agglomerative algorithms**: start with each point in a cluster and with clusters that are combined based on their "closeness", using one of many possible definitions of "close." (we have to define a distance measure)

- **Point assignment**: points are considered in some order, and each one is assigned to the cluster into which it best fits.

Hierarchical Clustering

We begin with every point in its own cluster. As time goes on, larger clusters will be constructed by combining two smaller clusters.



The output of hierarchical clustering is a dendogram, a diagram that shows the hierarchical relationship between objects. The main use of a dendrogram is to work out the best way to allocate objects to clusters. The key to interpreting a dendrogram is to focus on the height at which any two objects are joined together that indicates the order in which the clusters were joined. A more informative dendrogram can be created where the heights reflect the distance between the clusters.

3.5 Overfitting and Its Avoidance

Every model should have the property of **generalization**, refers to the ability of a trained model to perform well on new inputs unseen during training. Overfitting is the tendency of data mining procedures to tailor models to the training data, at the expense of generalization to previously unseen data points. All data mining procedures have the tendency to overfit so is fundamental manage the trade-off between model complexity and the possibility of overfitting.

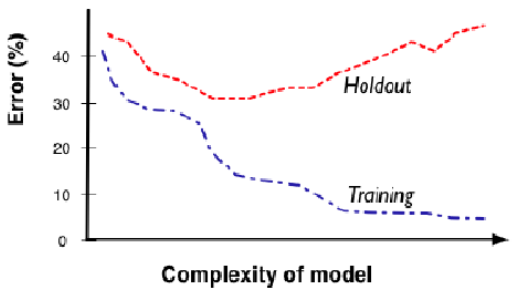
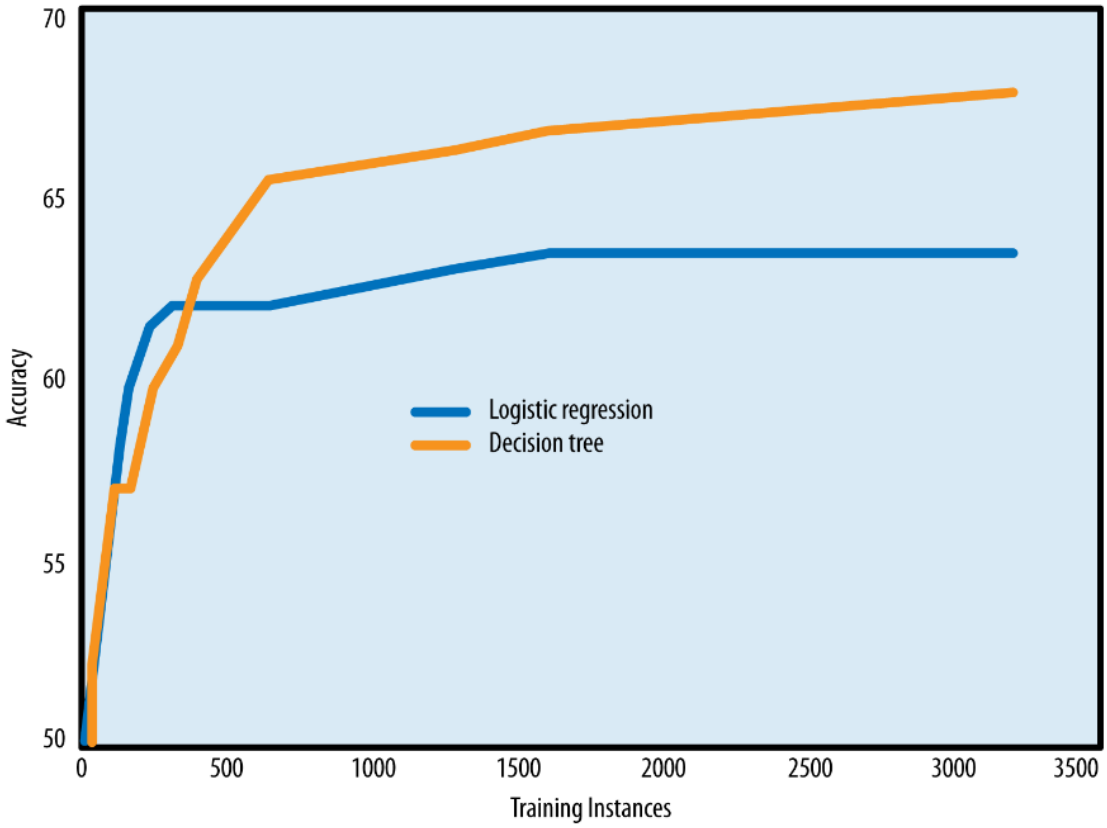


Figure 3.2: A fitting graph shows the accuracy of a model as a function of complexity

Evaluation on training data provides no assessment of how well the model generalizes to unseen

cases so, it's useful, creating holdout (test/validation) data as "lab test" of generalization performance, estimated comparing the predicted values with the true values.
Another graph to evaluate generalization performance is the **learning curve**. The learning curve may show that generalization performance has leveled off so investing in more training data is probably not worthwhile.



3.6 Evaluation

How predictive is the model we learned? Simple solution that can be used if lots of (labeled) data is split data into training and test set, an independent instances that have played no part in formation of classifier. The test data can't be used for parameter tuning and so proper procedure uses three sets: training data, test data, and validation data, used to optimize parameters. Generally, the larger the training data the better the classifier and the larger the test data the more accurate the error estimate.
How can we split train-test data? A common technique is the **cross-validation**: split data into k subsets of equal size and use each subset in turn for testing, the remainder for training. For these is also called **k-fold cross-validation**, where K is the number of subsets:

- **Stratified ten-fold cross-validation**: the folds are selected so that the mean response value is approximately equal in all the folds (useful for unbalanced dataset).
- **Leave-One-Out**: one instance as test and all the remaining instances as training (useful with small dataset).

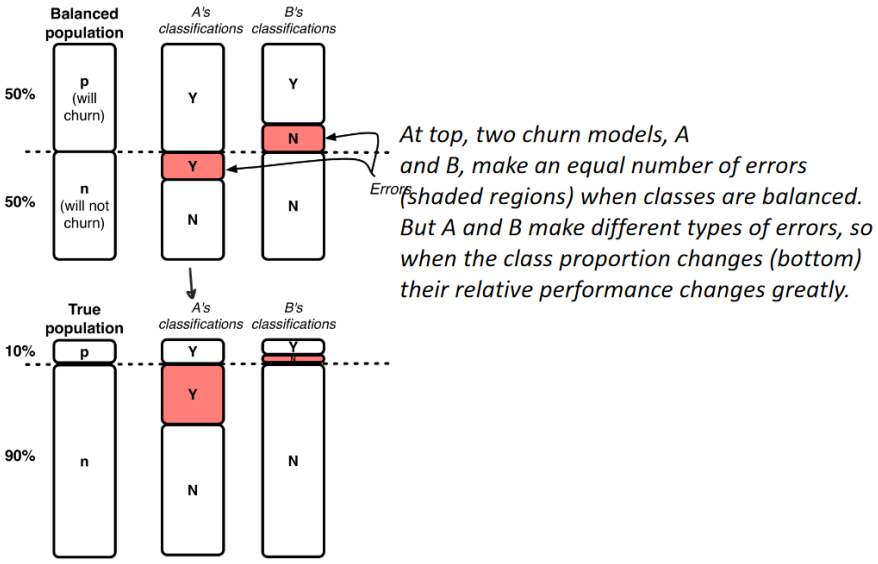
After the training we have to evaluate the model with some measures. An effective way is to use confusion matrix and define from that some measures as the following:

- **Accuracy** = (TP + TN) / (TP + TN + FP + FN)

- **TP rate** = TP / (TP+FN)
- **FP rate** = FP / (FP+TN)

		Predicted class	
		Yes	No
Actual class	Yes	True positive	False negative
	No	False positive	True negative

Unfortunately, as the class distribution becomes more skewed, evaluation based on accuracy breaks down. Even when the skew is not so great, in domains where one class is more prevalent than another accuracy can be greatly misleading. So we need to define other measures:

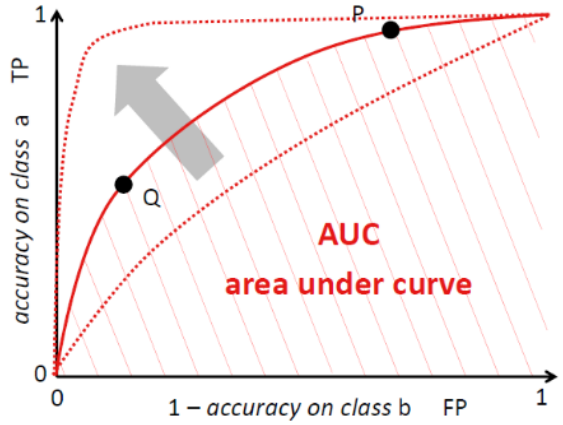


- **Recall** = TP / (TP+FN)
- **Precision** = TP / (TP+FP)

These two measures are in contrast between us so it's difficult to evaluate the model. Instead, we can use **F-measure** that combines precision and recall into a single metric.

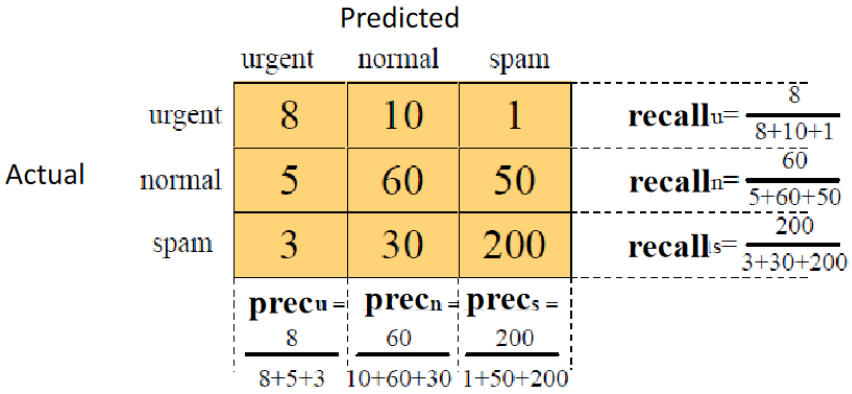
$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R} \tag{3.5}$$

Values of $\beta > 1$ favor recall, while $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced. If the model predicts as positive a single positive sample, the model precision on the positive class is 100%; lowering the threshold of a classifier may increase the denominator, increasing the number of results returned. For these reasons precision and recall are not absolute and they depend on threshold and application.
To understand what is the best value threshold (working point) we can use the AUC curve that measure the correlation between TP rate and FP rate.



Multi-label classification

We build separate binary classifiers for each class c.



For evaluation we can combine the values in two ways:

1. **Macroaveraging**, we compute the performance for each class, and then average over classes.
2. **Microaveraging**, we collect the decisions for all classes into a single contingency table, and then compute precision and recall from that table.

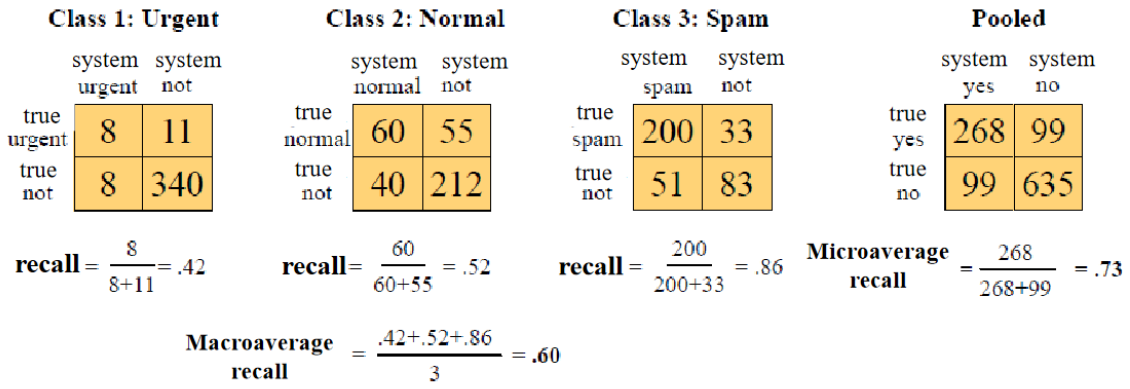


Figure 3.3: The best measure depend on the domain of model

3.7 Ensemble learning

Ensemble learning is a technique in machine learning where multiple models, often referred to as "base learners," are trained to solve the same problem and are combined to improve the overall performance. The primary objective of ensemble learning is to enhance the predictive accuracy and robustness of the model.

- **Bagging**: Involves training multiple versions of the same model on different subsets of the training data, obtained by sampling with replacement. The final prediction is usually made by averaging the predictions (for regression) or taking a majority vote (for classification). Random forests are a popular example of bagging applied to decision trees.
- **Boosting**: Sequentially trains models, each focusing on the mistakes made by the previous ones. The predictions are combined by weighting them according to the accuracy of the models. AdaBoost and Gradient Boosting are well-known boosting methods.

By combining multiple models, the ensemble can often achieve higher accuracy than any single model and can help in reducing overfitting by averaging out the errors of individual models.

4. Frequent Itemsets

This chapter delves into methods for identifying frequent itemsets in large datasets. The goal is to find efficient techniques to determine which items frequently appear together in transactions.

4.1 The Market-Basket Model

The market-basket model is used to describe transactions (baskets) containing sets of items and to discover patterns such as sets of items that frequently appear together. We assume that the number of items in a basket is small, much smaller, than the total number of items and that the number of baskets is very large, bigger than what can fit in main memory.

Definition 4.1.1 — Frequent Itemsets. A set of items is considered frequent if it appears in at least a specified number of baskets. The support threshold (s) is the minimum number of baskets required for the itemset to be considered frequent. Instead, the support for I is the number of baskets for which I is a subset.

$$\text{Support}(I) = \frac{\text{Number of baskets containing } I}{\text{Total number of baskets}} \tag{4.1}$$

Items = {milk, coke, pepsi, beer, juice}

Minimum support = 3 baskets

B1 = {m, c, b}	B2 = {m, p, j}
B3 = {m, b}	B4= {c, j}
B5 = {m, p, b}	B6 = {m, c, b, j}
B7 = {c, b, j}	B8 = {b, c}

Frequent itemsets: {m}, {c}, {b}, {j}, {m,b}, {b,c}, {c,j}

Frequent itemset analysis has applications in various domains, including:

- **Supermarkets**: items are products that the store sells and baskets are set of items bought together. Finding frequent itemsets, a retailer can learn what is commonly bought together.
- **Related Concepts**: items are word and baskets are documents. The goal is find, among the frequent pairs, some pairs of words that represent a joint concept.
- **Plagiarism**: items are documents and baskets are sentences. We look for pairs of items that appear together in several baskets because, in that case, two documents share several sentences in common.

4.1.1 Association Rules

Association rules are if-then statements that help uncover relationships between itemsets. For example, if a basket contains items A and B, it is likely to contain item C (A,B $\rightarrow$ C). In practice there are many rules, we want to find significant ones; to do that we can use some metrics:

- **Confidence**: The probability that a basket containing itemset I also contains item j.

$$\text{Confidence}(I \rightarrow j) = \frac{\text{Support}(I \cup j)}{\text{Support}(I)} \tag{4.2}$$

  Using only confidence is useless because, for example, we have the rule X $\rightarrow$ milk may have high confidence for many itemsets X, because milk is purchased very often (independent of X) and the confidence will be high.
- **Interest**: Difference between its confidence and the probability of find j in the baskets.

$$\text{Interest}(I \rightarrow j) = \text{Confidence}(I \rightarrow j) - \text{Support}(j) \tag{4.3}$$

  Good association rules have confidence high and interest low.

So the problem is to find all association rules with support $\geq$ s and confidence $\geq$c. The procedure is a two-steps algorithm:

- Finding the Frequent Itemsets (hardest part of the work)
- Generating the Association Rules

4.1.2 Representation of Market-Basket Data

Before compute the algorithm we have to understand what is the best representation of market-basket data. Typically market basket data is stored in a file basket-by-basket; if the size of the file is large it does not fit in main memory. So major cost for any algorithm is the cost to read baskets from the file.

Once a disk block full of baskets is in main memory, we can expand it, generating all the subsets of size k. Time to generate all the subsets of size k for a basket with n items could dominate the time needed to transfer the data from disk; it is usually possible to eliminate many of the items in each basket as not able to participate in a frequent itemset, so the value of n drops as k increases. So we can measure the running time of a frequent-itemset algorithm by the number of times that each disk block of the data file is read.

All frequent-itemset algorithms require us to maintain many different counts as we make a pass through the data, for example we need to count the number of times that each pair of items occurs in baskets. If we do not have enough main memory to store each of the counts, then each operation will require us to load a page from disk. There are two main representation of market-basket data:

- **Triangular Matrix**: A space-efficient way is to use a one-dimensional triangular array. We store in $a[k]$ the count for the pair i, j, with $1 \leq i < j \leq n$, where:

$$k = (i-1)(n - \frac{i}{2}) + j - 1 \tag{4.4}$$

  The result of this layout is that the pairs are stored in lexicographic order.

- **The Triples**: counts stored as triples $[i, j, c]$ where the count of pair i, j, with $i < j$, is c. A data structure, such as a hash table with i and j as the search key, is used so we can tell if there is a triple for a given i and j and, if so, to find it quickly. The triples method requires us to store three integers, rather than one, for every pair that does appear in some basket.

The triangular matrix is better if at least 1/3 of the possible pairs actually appear in some basket. Instead, the triples method is better if significantly fewer than 1/3 of the possible pairs occurs.

4.1.3 Monotonicity of Itemsets

An important propriety that every algorithm use to compute frequency of itemsets is the **monotonicity** property.

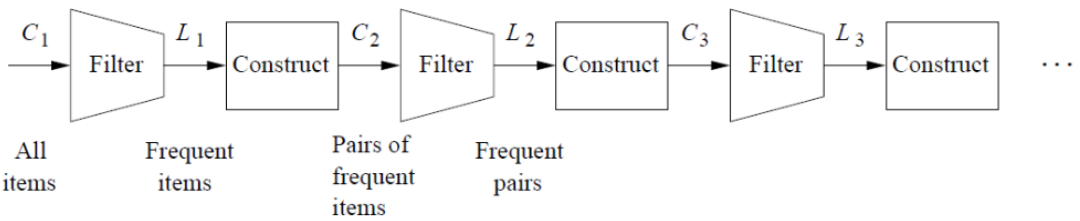
Definition 4.1.2 — Monotonicity of Itemsets. If a set I of items is frequent, then so is every subset of I.

In practice the most main memory is required for determining the frequent pairs. In order for frequent-itemset analysis to make sense, the result has to be a small number of sets. In practice the support threshold is set high enough that it is only a rare set that is frequent and, thanke to monotonicity, if there is a frequent triple, then there are three frequent pairs contained within it. Thus, we expect to find more frequent pairs than frequent triples, more frequent triples than frequent quadruples, and so on.

4.2 The A-Priori Algorithm

The A-Priori Algorithm is designed to reduce the number of pairs that must be counted, at the expense of performing two passes over data.

1. In the **first pass**, we create two tables:
   (a) The first table translates item names into integers from 1 to n
   (b) The other table is an array of counts; the ith array element counts the occurrences of the item numbered i
2. **Between the passes**, we examine the counts of the items to determine which of them are frequent as singletons. This table, called **frequent-items table** is an array indexed 1 to n, and the entry for i is either 0, if item i is not frequent, or a unique integer in the range 1 to m if item i is frequent.
3. In the **second pass**, for each basket, look in the frequent-items table to see which of its items are frequent. In a double loop, generate all frequent pairs and for each frequent pair, add one to its count in the data structure used to store counts. Finally, at the end of the second pass, examine the structure of counts to determine which pairs are frequent.



We start with C_1, which is all singleton itemsets. The first filter step is to count all items, and those whose counts are at least the support threshold s form the set L_1 of frequent items. The set C2 of candidate pairs is the set of pairs both of whose items are in L_1. The second pass of the A-Priori Algorithm counts all the candidate pairs and determines which appear at lest s times. These pairs

form L_2, the frequent pairs. We can follow this path as far as we wish.
The A-Priori Algorithm is fine as long as the step with the greatest requirement for main memory has enough memory that it can be accomplished without thrashing.

4.3 The PCY Algorithm

The PCY (Park-Chen-Yu) Algorithm enhances the A-Priori Algorithm by using a hash table during the first pass to count item pairs and filter out infrequent pairs. It exploits the observation that there may be much unused space in main memory on the first pass.

1. **First Pass**: Add 1 to the count for each item in the basket, generate all the pairs, using a double loop and hash each pair, adding 1 to the bucket into which that pair hashes. Between the passes of PCY, the hash table is summarized as a bitmap, with one bit for each bucket.
2. **Second Pass**: At the end of the first pass, each bucket has a count. If the count is at least as great as the support threshold s, it is a frequent bucket. In the case of an infrequent bucket, no pair that hashes to this bucket can be frequent.

We can define the set of candidate pairs C_2 to be those pairs i, j such that:

- i and j are frequent items
- i, j hashes to a frequent bucket

It is the second condition that distinguishes PCY from A-Priori. Another difference is that , while we were able to use the triangular-matrix method on the second pass of A-Priori, because the frequent items could be renumbered from 1 to some m, we cannot do so for PCY. The reason is that the pairs of frequent items that PCY lets us avoid counting are placed randomly within the triangular matrix. There is no known way of compacting the matrix to avoid leaving space for the uncounted pairs and, consequently, we are forced to use the triples method in PCY.

4.3.1 Randomized Algorithms

There are applications where it is not essential to discover every frequent itemset. For example, if we are looking for items purchased together at a supermarket, we are not going to run a sale based on every frequent itemset we find.
We pick a random subset of the baskets and pretend it is the entire dataset. We must adjust the support threshold to reflect the smaller number of baskets; if the support threshold for the dataset is s, and we choose a sample of 1% of the baskets, we should examine the sample for itemsets that appear in at least s/100 of the baskets. The safest way to pick the sample is to read the entire dataset, and select a basket with some fixed probability p. Then we execute one of the previous algorithms on the sample and we extract the frequent itemsets.

1. **Sampling**: Select a representative sample of the data.
2. **Finding Frequent Itemsets**: Apply the A-Priori Algorithm to the sample.
3. **Validation**: Verify the frequent itemsets in the entire dataset.

If the sample is large enough, there are unlikely to be serious errors:

- An itemset that is frequent in the whole but not in the sample is a false negative
- An itemset that is frequent in the sample but not the whole is a false positive

We can eliminate false positives by making a pass though the full dataset and counting all the itemsets that were identified as frequent in the sample; another option can be execute the algorithm on different portion of dataset and take elements in common. These improvements will eliminate all false positives, but a false negative is not counted and therefore remains undiscovered. To reduce the number of false negatives adding more memory or reducing the threshold.

4.3.2 The SON Algorithm

The SON (Savasere-Omiecinski-Navathe) Algorithm is designed for distributed computing environments like MapReduce. The idea is to divide the input file into chunks and treat each chunk as a sample, and run the algorithm. We use ps as the threshold, if each chunk is fraction p of the whole file, and s is the support threshold. Store on disk all the frequent itemsets found for each chunk and take the union of all the itemsets that have been found frequent for one or more chunks. In a second pass, we count all the candidate itemsets and select those that have support at least s as the frequent itemsets.
If an itemset is not frequent in any chunk, its support is less than ps in each chunk and since the number of chunks is 1/p, the total support for that itemset is less than (1/p)ps = s. So every itemset that is frequent in the whole is frequent in at least one chunk, and there are no false negatives.

4.4 Toivonen's Algorithm

The algorithm uses a pass over a sample and a full pass over the data. Toivonen's algorithm begins by selecting a small sample of the input dataset, and finding from it the candidate frequent itemsets. Then it constructs the **negative border**, a collection of itemsets that are not frequent in the sample, but all of their immediate subsets are frequent in the sample.

Items: {A,B,C,D,E}

We know that frequent itemsets are {A}, {B}, {C}, {D}, {B,C},{C,D}

{E} is in the negative border, because it is not frequent in the sample, but its only immediate subset, ∅, is frequent

The sets {A,B}, {A,C}, {A,D} and {B,D} are in the negative border

None of the triples or larger sets are in the negative border

Then we make a pass through the entire dataset, counting all the itemsets that are frequent in the sample or are in the negative border. Two possible outcomes:

1. No member of the negative border is frequent in the whole dataset. In this case, the correct set of frequent itemsets is exactly those itemsets from the sample that were found to be frequent in the whole.
2. Some member of the negative border is frequent in the whole and we cannot be sure that there are not some even larger sets. We can give no answer at this time and must repeat the algorithm with a new random sample.

5. Finding Similar Items

This chapter delves into methods for identifying similar items in large datasets, a fundamental task in data mining. The goal is to find efficient techniques to determine which items are similar based on specific criteria like textual content or user behavior. Near-neighbor search is essential for identifying items that are similar to each other. This section discusses several applications where finding similar items is crucial:

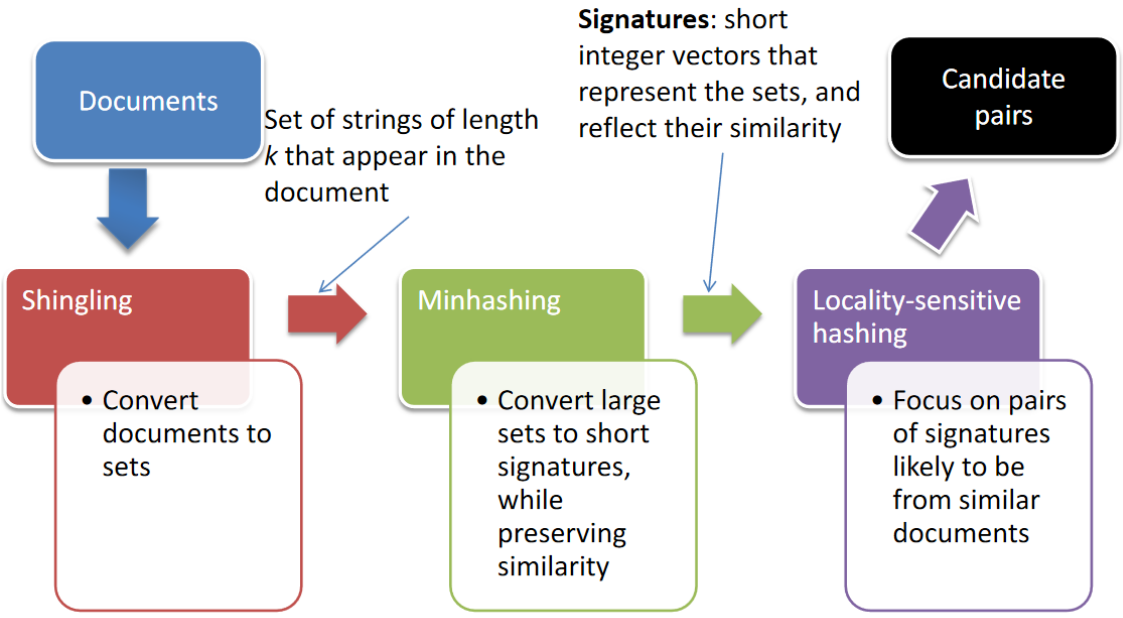
- **Textual Similarity of Documents**: This involves identifying near-duplicate web pages or documents using the **Jaccard similarity** metric. This metric measures the size of the intersection divided by the size of the union of two sets.

$$J(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \tag{5.1}$$

- **Plagiarism Detection**: Finding plagiarized documents by examining character-level similarity. Plagiarized content may share significant portions of text with the original, despite minor alterations.
- **Mirror Pages**: Detecting mirror web pages that are nearly identical but hosted on different servers.
- **Articles from the Same Source**: Identifying news articles from the same source but with slight modifications. News aggregators, like Google News, use this to avoid showing multiple versions of the same article.
- **Collaborative Filtering**: Recommending items by finding users or products with similar behavior or features. For instance, Amazon and Netflix use this method to suggest products and movies based on past user behavior.

5.1 Similarity of Documents

The essential steps for compute similarity between documents are shown in the figure below.



5.1.1 Shingling of Documents

The most effective way to represent documents as sets is to construct the set of short strings that appear within it. Define a k-shingle for a document to be any substring of length k found within the document and associate with each document the set of k-shingles that appear one or more times within that document. For example, for a document "abcdabd" with $k = 2$, the 2-shingles are$\{ab, bc, cd, da, bd\}$.
Choosing the appropriate size k for shingles is crucial. If k is too small, most documents will appear similar. If k is too large, common phrases might be missed and computation can be heavy. Typically, k is chosen based on the length and typical content of the documents being compared (k should be picked large enough that the probability of any given shingle in any given document is low).

5.1.2 Hashing Shingles

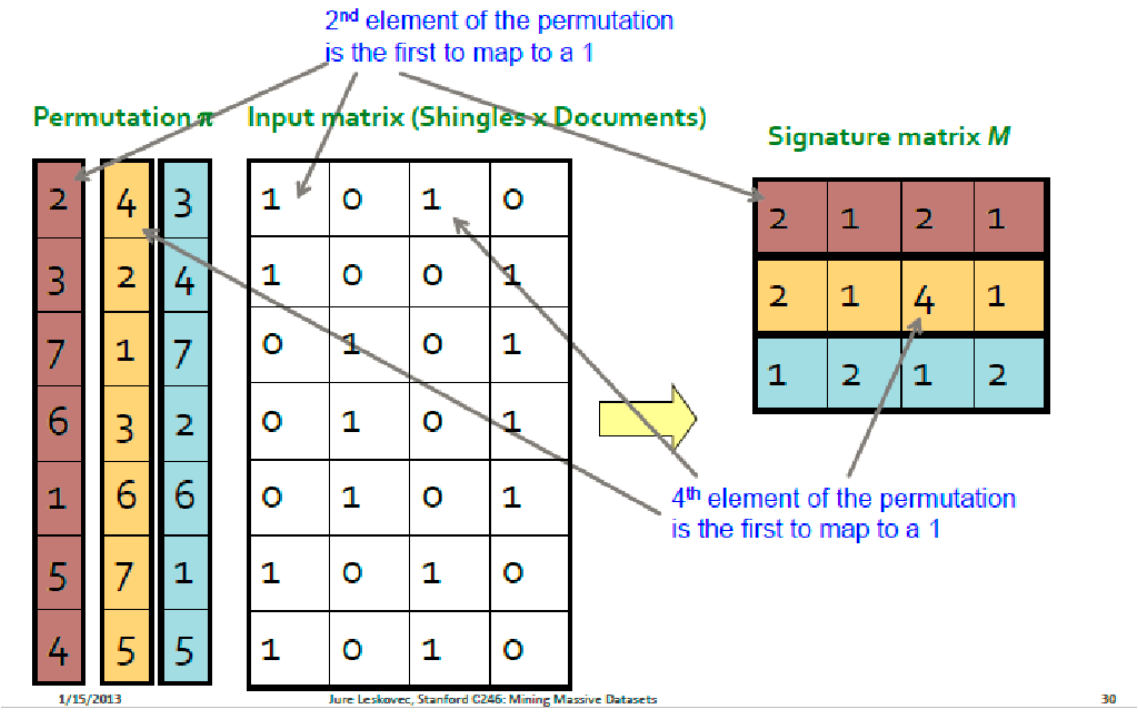
Instead of using the actual shingles, a hash function can be used to map shingles to integer values, reducing the data size while preserving the ability to compare documents effectively. Suppose we need to find near-duplicate documents among N=1 million documents. Naïvely, we'd have to compute pairwaise Jaccard similarites for every pair of docs; for N = 10 million, it takes more than a year. So Our goal is to replace large sets by much smaller representations called "signature. The important property we need for signatures is that we can compare the signatures of two sets and estimate the Jaccard similarity of the underlying sets from the signatures alone. Obviously, it is not possible that the signatures give the exact similarity of the sets they represent, but the estimates they provide are close.
The key idea is to "hash" each column C to a small signature h(C), such that h(C) is small enough that the signature fits in RAM and the sim(C1, C2) is the same as the "similarity" of signatures h(C1) and h(C2). So the goal is to find a right hash function h(·):

- **From bits to decimal**: Take group of 3/4 bits and hash in their decimal representation. This method doesn't work because produce few elements in common.
- **Module**: Also this method doesn't work because there are too many collision (1001, 2001, 3001, 4001 collide all in the same hash).

- **Minhashing**: Compresses sets into small signatures while preserving their Jaccard similarity.

Minhashing

A minhash function is defined by permuting the rows of the characteristic matrix and using the position of the first 1 in each column as the minhash value.



There is a connection between minhashing and Jaccard similarity of the sets that are minhashed. The probability that the minhash function for a random permutation of rows produces the same value for two sets equals the Jaccard similarity of those sets.

$$Pr[h(C1) = h(C2)] = sim(C1, C2) \tag{5.2}$$

How can we demonstrate that? Rows can be divided into three classes:

- Type X rows have 1 in both columns.
- Type Y rows have 1 in one of the columns and 0 in the other
- Type Z rows have 0 in both columns

Since the matrix is sparse, most rows are of type Z. Similary is equals to the number of rows of type X (intersection) divide by the sum between number of rows of type X and number of rows of type Y (union).

$$SIM(S1, S2) = \frac{S1 \cap S2}{S1 \cup S2} = \frac{x}{x+y} \tag{5.3}$$

Consider the probability that h(S1) = h(S2). Given the rows permuted randomly, and we proceed from the top, the probability that we meet a type X row before a type Y is x/(x + y).

- If the first (no Z) row from the top is a type X row, then surely h(S1) = h(S2).
- If the first (no Z) row from the top is a type Y row, the set with a 1 gets that row as its minhash value and the set with a 0 in that row surely gets some row further down the permuted list.

Thus, we know h(S1) $\neq$ h(S2) if we first meet a type Y row. So the probability that h(S1) = h(S2) is x/(x + y), which is the Jaccard similarity of S1 and S2.

Computing Minhash Signatures

In practice, explicit row permutations are replaced by using hash functions to simulate the effect of permutations efficiently. We pick n randomly chosen hash functions $h1, h2, \ldots, h_n$ and we construct the signature matrix by considering each row in their given order.

$$h_{a,b}(x) = ((ax+b) mod\, p) mod N \tag{5.4}$$

To understand better the procedure consider the example shown in the figure below.

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

The first step of the algorithm for computing the signature matrix is shown in the figure below.

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Start from row 0: the values of $h_1(0)$ and $h_2(0)$ are 1. The row numbered 0 has 1's in the columns for sets S_1 and S_4, so only these columns of the signature matrix can change. As 1 is less than inf, we change both values in the columns for S_1 and S_4.

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Then row 1. This row has 1 only in S_3, and its hash values are h1(1) = 2 and h2(1) = 4. Thus, SIG(1, 3) to 2 and SIG(2, 3) to 4

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

Row 2 has 1 in S_2 and S_4, with values h1(2) = 3 and h2(2) = 2. We don't change the values for S_4, since the values in the matrix, [1, 1], are each less than the hash values [3, 2]. Since S2 has inf, we replace it by [3, 2].

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

Row 3 has all columns but S_2 with 1, and the hash values are h1(3) = 4 and h2(3) = 0.The value 4 for h_1 exceeds what is in the matrix for all the columns, so we shall not change any values in the first row. However, the value 0 for h_2 is less than what is present, so we lower SIG(2, 1), SIG(2, 3) and SIG(2, 4) to 0.

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

Finally row 4 that has h1(4) = 0 and h2(4) = 3. Since row 4 has 1 only in the column for S_3, we only compare the signature column for that set, [2, 0] with the hash values [0, 3]. We change only SIG(1, 3) to 0.

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

Speeding Up Minhashing

The process of minhashing is time-consuming, since we need to examine the entire k-row matrix M for each minhash function we want. In the original model, to compute one minhash function, we shall not go all the way to the end of the permutation but we have only look at the first m out of k rows. If we make m small compared with k, we reduce the work by a large factor, k/m. As long as each column has at least one 1 in the first m rows in permuted order, the rows after the mth have no effect on any minhash value.

When we examine the minhash signatures of two columns to estimate the Jaccard similarity of their underlying sets, there are three cases:

1. If neither column has inf in a given row, then there is no change needed.
2. One column has inf and the other does not. We surely have an example of unequal minhash values, and we count this row of the signature matrix as such an example.
3. Both columns have inf in row. We have no information about the Jaccard similarity of the corresponding sets; that similarity is only a function of the last k-m rows, which we have chosen not to look at. That effect will reduce the accuracy of our estimates of the Jaccard distance somewhat, but not much.

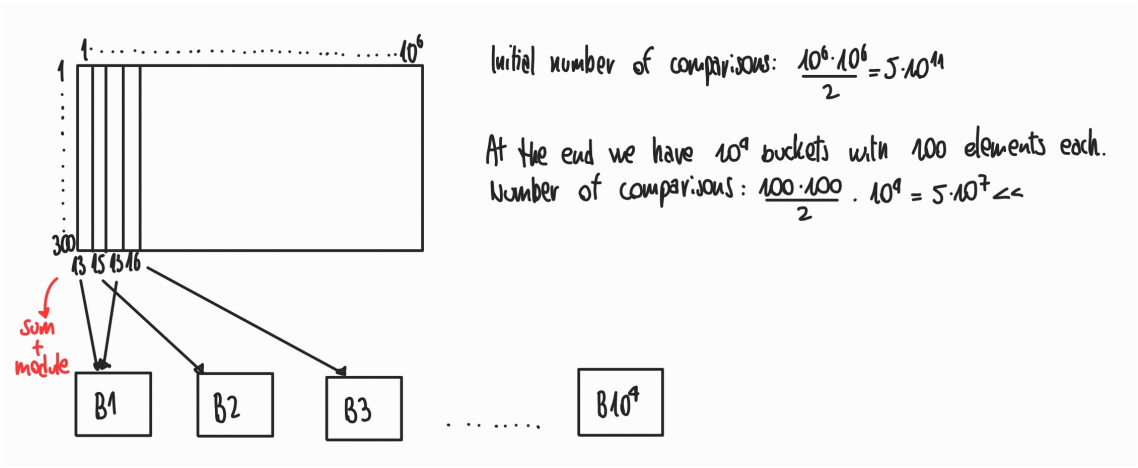
Since we are able to compute minhash values much faster, we can afford the time to apply a few more minhash functions. We get even better accuracy than originally, and we do so faster than before.

5.1.3 Locality-Sensitive Hashing for Documents

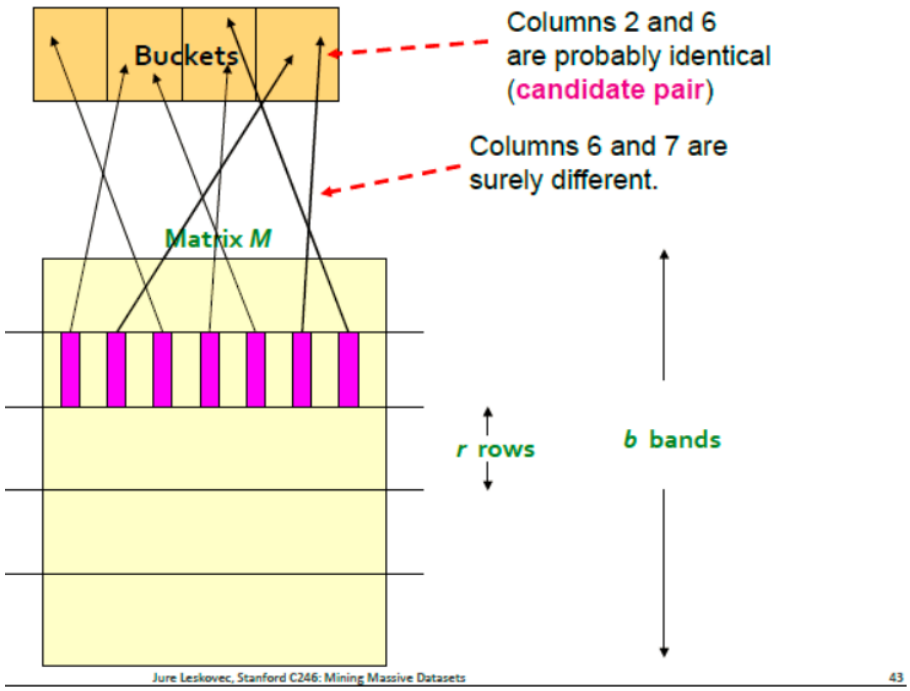
Locality-sensitive hashing (LSH) aims to reduce the number of comparisons needed to find similar items by focusing on pairs that are likely to be similar. The general idea is to use a function f(x,y) that tells if x and y is a candidate pair: a pair of elements whose similarity must be evaluated. With the minhash matrices, the idea is to hash columns of signature matrix M to many buckets. Each pair of documents that hashes into the same bucket is a candidate pair and the hope is that most of the dissimilar pairs will never hash to the same bucket, and therefore will never be checked.

One method is to add up the values in each column of the minhash matrices and then find the remainder after dividing by a specific number. This will assign each column to a bucket as shown in the figure below. While this significantly decreases the number of necessary comparisons, it also

results in a lot of false negatives. This is because even a small change in a column's value can cause it to end up in a different bucket.



An more effective method is divide matrix M into b bands of r rows and for each band, hash its portion of each column to a hash table with k buckets (k as large as possible). Candidate column pairs are those that hash to the same bucket for ≥ 1 band. Columns that do not agree in band 1 have other chances to become a candidate pair because they might be identical in any one of these other bands. And the end we make the comparison of documents that are in the same bucket, meaning what they have at least one band in common.



Analysis of the Banding Technique

Recall that the probability the minhash signatures for these documents agree in any one particular row of the signature matrix is s. We calculate the probability that these documents become a

candidate pair as follows:

- The probability that the signatures agree in all rows of one particular band is s^r.
- The probability that the signatures do not agree in at least one row of a particular band is $1-s^r$.
- The probability that the signatures do not agree in one rows of any of the bands is $(1-s^r)^b$.
- The probability that the signatures agree in all the rows of at least one band, and become a candidate pair, is $1-(1-s^r)^b$

To tune b and r to catch most similar pairs, we can use $1-(1-s^r)^b$ function that has the form of an S-curve.

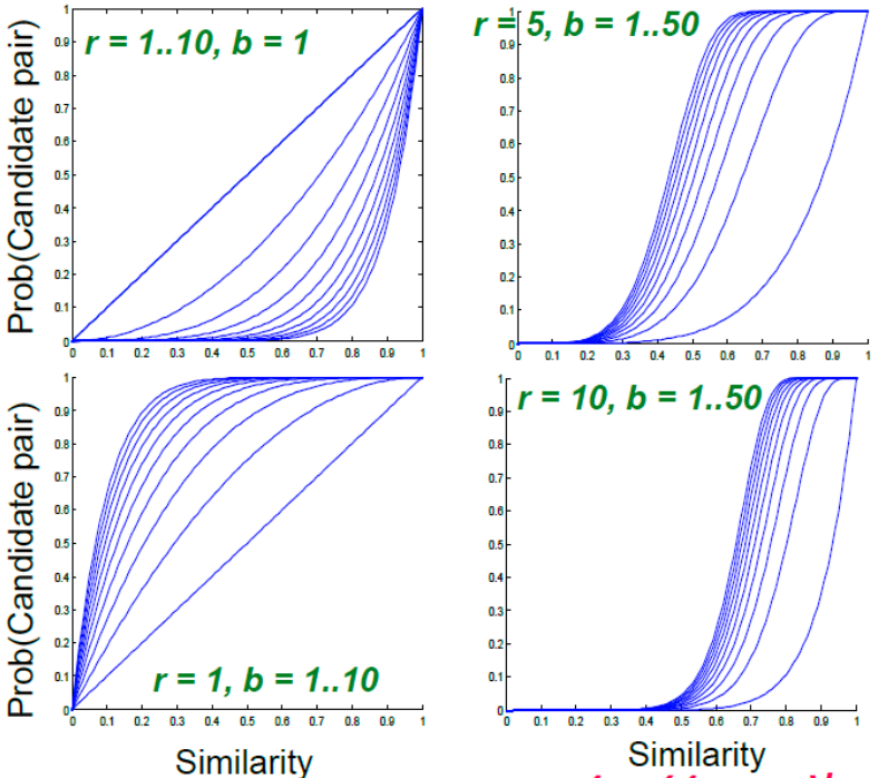


Figure 5.1: To balance false positives/negatives, given a fixed threshold s, we need to pick the right number of minhashes (rows of M), the number of bands b, and the number of rows r per band

Assume the following case: Suppose 100,000 columns of M (100k docs), signatures of 100 integers (rows) and choose b = 20 bands of r = 5 integers/band.

1. Assume: $sim(C1,C2) = 0.8$. Since sim(C1, C2) $\geq$ s, we want C1, C2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)
   - Probability C1, C2 identical in one particular band: $(0.8)^5 = 0.328$
   - Probability C1, C2 are not similar in all of the 20 bands: $(1-0.328)^{20}$ = 0.00035
   - We would find 99.965% pairs of truly similar documents
2. Assume: $sim(C1,C2) = 0.3$. Since sim(C1, C2) < s we want C1, C2 to hash to NO common buckets (all bands should be different).
   - Probability C1, C2 identical in one particular band: $(0.3)^5 = 0.00243$.
   - Probability C1, C2 identical in at least 1 of 20 bands: $1-(1-0.00243)^{20} = 0.0474$.
   - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.

II Text Analysis

6. Text Retrieval

The fundamental objective of text data access is to connect users efficiently with the information they need at the right time. This can be achieved through two primary methods:

- In the **"pull"** method, users actively search for information, taking the initiative to fetch relevant data from the system (search engine).
- Conversely, the **"push"** method involves the system proactively suggesting information to users based on their potential needs (recommendation system).

6.1 Text Retrieval (TR)

Text Retrieval (TR) systems help users find relevant documents based on their queries. This task is challenging due to often incomplete user queries and the difficulty in precisely describing information needs. Furthermore, determining what constitutes a relevant document can be highly subjective, with human experts sometimes disagreeing on the correct answer. TR differs from traditional database retrieval as it deals with unstructured text and keyword-based, often vague queries, whereas databases handle structured data with well-defined schemas and structured queries. In a more formal way, given a document collection, the task of text retrieval can be defined as using a user querto identify a subset of documents that can satisfy the user's information need.
Let $V = \{w_1, \ldots, w_N\}$ be a vocabulary set of all the words in a particular natural language where w_i is a word. A user's query $q = q_1, q_2, \ldots, q_m$ is a sequence of words, where $q_i \in V$. A document $di = d_{i1}, \ldots, d_{is}$ is a sequence of words where $d_{ij} \in V$. Our text collection $C = \{d_1, \ldots, d_M\}$ is a set of text document. We assume that there exists a subset of documents in the collection $R(q) \subset C$, which are relevant to the user's query q. Different users may use the same query to intend to retrieve different sets of relevant documents, so it is unrealistic to expect a computer to return exactly the set R(q). Thus, the best a computer can do is to return an approximation of R(q), which we will denote by R'(q).
Now, how can a computer compute R'(q)?
In a high level, there are two alternative strategies:

- **Document selection** employs binary classification to determine the relevance of documents,

essentially categorizing them as either relevant or non-relevant.

$$R'(q) = \{d | f(q,d) = 1, d \in C\} \tag{6.1}$$

- **Document ranking**, on the other hand, involves ranking documents by their relevance to the query, allowing users to set their own thresholds for what constitutes a relevant result. Estimation of relative relevance is easier than that of absolute relevance.

$$R'(q) = \{d | f(q,d) \geq \theta\} \tag{6.2}$$

Ranking is generally preferred to document selection for multiple reasons; the main is described by the following theorem.

Theorem 6.1.1 — Probability ranking principle. The strategy of ranking is further shown to be optimal theoretically under two assumptions based on the probability ranking principle, which states that returning a ranked list of documents in descending order of predicted relevance is the optimal strategy under the following two assumptions.

- The utility of a document to a user is independent of the utility of any other document
- A user will browse the results sequentially

6.1.1 Retrieval Models

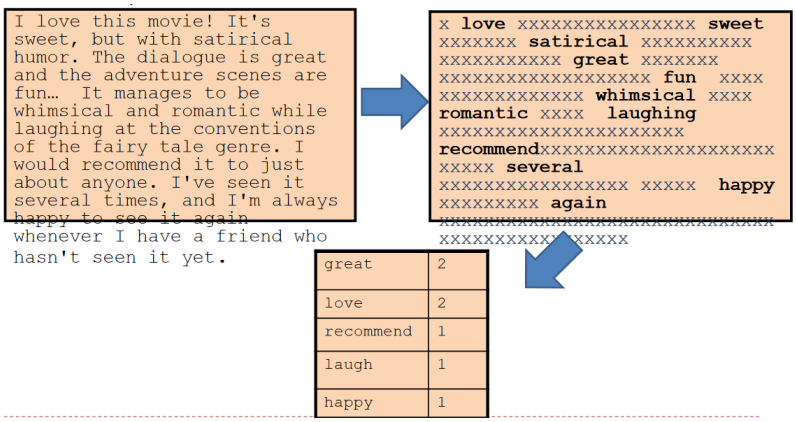
So the problem is the following: we have a query and a document, and we want to define the function $f(.,.)$ that computes a score based on the query and document. Our function has to measure the likelihood that a document d is relevant to a query q.
We achieve this goal by designing a retrieval model which gives us a formalization of relevance. Retrieval models can be broadly categorized into similarity-based models and probabilistic models.

- **Similarity-based models**, such as the Vector Space Model (VSM), rank documents based on their similarity to the query.
- On the other hand, **probabilistic retrieval models** estimate the likelihood of a document being relevant to a query.

Most of models are all based on the assumption of using a **bag-of-words** representation of text that define three measures to characterize the popularity of the term in the collection:

1. **Term frequency (TF)**: number of times that a word occurs in the document d.
2. **Document Length**.
3. **Document frequency (DF)**: number of times that a word occurs in the entire collection.

Any models are equally effective but we don't have a single winner. The state-of-the-art ranking functions tend to rely to the bag of words representation, the TF and the document frequency of words.



Vector Space Models (VSM)

The vector space (VS) retrieval model is a simple, yet effective method of designing ranking functions. It is a special case of similarity-based models, where we assume relevance is correlated to similarity between a document and a query. To compute similarity documents and queries are represented as vectors where each dimension corresponds to a term. A term can be any basic concept such as a word or a phrase, or even n-grams of characters or any other feature representation. Each term is assumed to define one dimension; therefore, since we have $|V|$ terms in our vocabulary, we define a $|V|$-dimensional space. A query vector consists of a number of elements corresponding to the weights of terms.
The simplest instantiation of the Vector Space Model is the following:

$$q = (x_1, \cdots x_N) \qquad x_i, y_i \in \{0, 1\}$$
$$\textbf{1: word } W_i \textbf{ is present}$$
$$d = (y_1, \cdots y_N) \qquad \textbf{0: word } W_i \textbf{ is absent}$$

$$Sim(q, d) = q.d = x_1 y_1 + \cdots + x_N y_N = \Sigma_{i=1}^{N} x_i y_i$$

As shown in example below, there are three documents, d2, d3, and d4, that have a score of 3. But d4 should be right above d3 since d3 only mentioned presidential once while d4 mentioned it many more times. d2 and d3 also have the same score, but intuitively, d3 is more relevant and should be scored higher than d2.

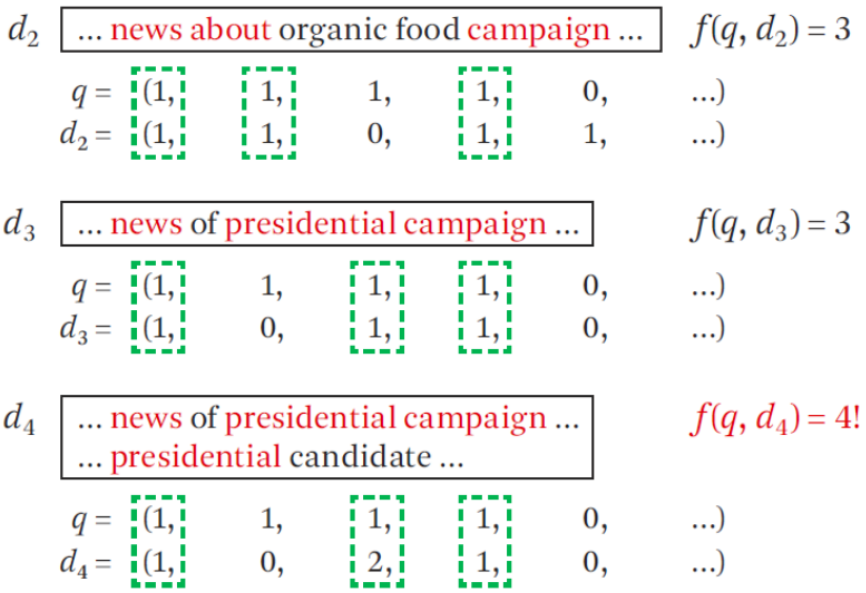
Query = "news about presidential campaign"

d1	… **news about** …	f(q,d1)=2
d2	… **news about** organic food **campaign**…	f(q,d2)=3
d3	… **news** of **presidential campaign** …	f(q,d3)=3
d4	… **news** of **presidential campaign** … … **presidential** candidate …	f(q,d4)=3
d5	… **news** of organic food **campaign**… **campaign**…**campaign**…**campaign**…	f(q,d5)=2

This model doesn't do that, and we have to solve these problems. A natural thought is to consider multiple occurrences of a term, term frequency (TF), in a document as opposed to binary representation.

$$TF(w,d) = count(w,d) \tag{6.3}$$

The problem with the previous rank is that d2 and d3 still have identical scores. Is there any way to determine which word should be treated more importantly and which word can be essentially ignored? About doesn't carry much content, so we should ignore it. This word are called **stop word** and are generally very frequent and have any significance.

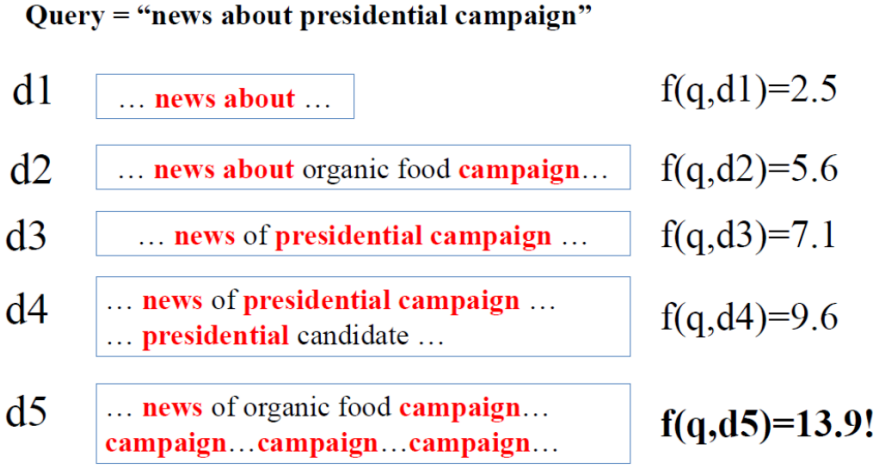


We could somehow use the global statistics to decrease the weight of the stop words and at the same time to increase the weight of the other words. This idea is called the **inverse document frequency (IDF)**. The document frequency is the count of documents that contain a particular term; inverse because we want to penalize a word occurring in many documents and reward informative words that have a higher IDF.

$$IDF(w) = (\frac{M+1}{df(w)}) \tag{6.4}$$

where M is the total number of documents in the collection and df(w) counts the document frequency. Adding IDF to the model, the formula become the following.

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t) \tag{6.5}$$

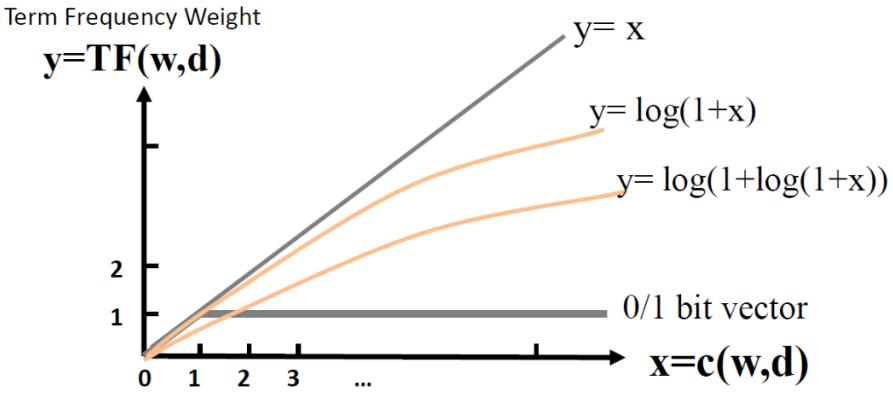


The presented model actually works pretty well for the examples shown in the figures, except for d5 which has a very high score. This document is intuitively non-relevant, so its position is not

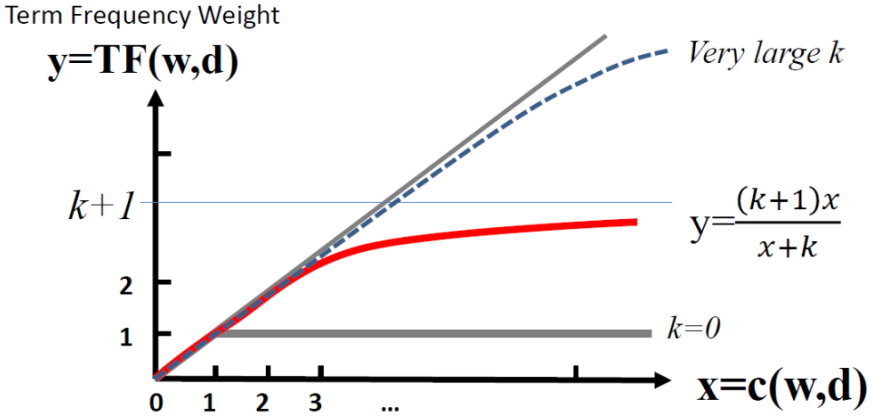
desirable. To make penalization of IDF less important in the computation of score we can use various transformation using the logarithm.

$$f(q,d) = \sum_{w \in q \cap d} c(w,q) TF(w,d) \log \frac{M+1}{df(w)} \tag{6.6}$$

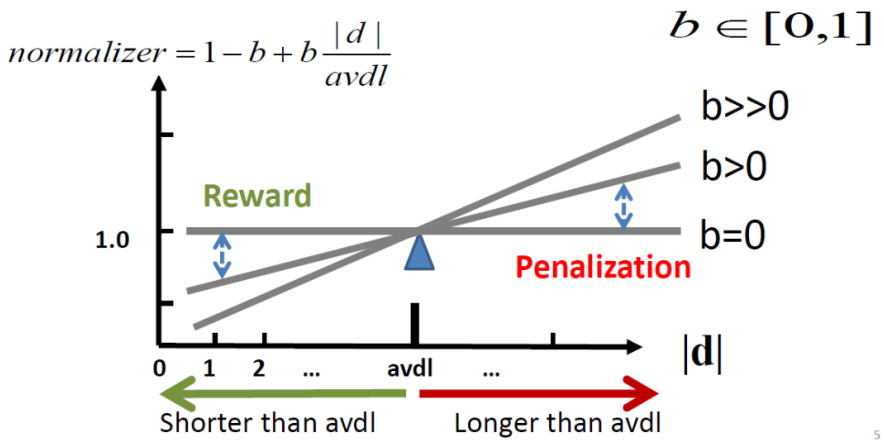
Once we see a word in the document, it's very likely that the document is talking about this word. An extra occurrence on top of the first occurrence, we also can say the second occurrence confirmed that it's not a accidental mention of the word. A big amount of occurrences of the word in the document is not going to bring new evidence about the term because we are already sure that this document is about this word. So we can make some transformation to TF factor to give it less weight in the formula.



A very common transformation is the **BM25 transformation**, that uses the factor shown in the figure below. It's easy to see this function has a upper bound; where k = 0, we have a zero one bit transformation. Instead, if we set k to a very large number, it's going to look more like the linear transformation function.



Long documents would have a higher chance to match a query since they contain more content. In this sense, we should penalize long documents because they have a better chance to match a query and need to be careful not to over-penalize small documents. So we need to be careful about using the right degree of length penalization, and an understanding of the discourse structure of documents is needed for optimal document length normalization. A common normalization is the**Pivoted Length Normalization**.



The idea is to use the average document length as a pivot. If a document is longer than the average length, we apply some penalization; if it's shorter, there's even some reward. The formula for the normalizer is an interpolation of 1 and the normalized document lengths, controlled by a parameter b. If we set the parameter b to zero, then the normalizer value would be one, indicating no length normalization at all. Instead, if we set b to a nonzero value, then the value would be higher for documents that are longer than the average document length, whereas the value of the normalizer will be smaller for shorter documents.
Popular ranking functions are the following:

- Pivoted Length Normalization VSM

$$f(q,d) = \sum_{w \in q \cap d} c(w,q) \frac{\ln[1+\ln[1+c(w,d)]]}{1-b+b\frac{|d|}{avdl}} \log \frac{M+1}{df(w)} \tag{6.7}$$

- Okapi BM25

$$f(q,d) = \sum_{w \in q \cap d} c(w,q) \frac{(k+1)c(w,d)}{c(w,d)+k(1-b+b\frac{|d|}{avdl})} \log \frac{M+1}{df(w)} \tag{6.8}$$

Probabilistic Retrieval Models

We introduce a binary random variable R and we model the query and the documents as observations from random variables.
In a first approach, we can compute the probability, relevance judgements for queries and documents, just counting the positive results. The main problem is that we don't observe all the queries and all of the documents and all the relevance values; there will be many unseen documents (we can only collect data from the documents that we have shown to the users).
What do we do when we have a lot of unseen documents and queries? The solution is that we have to approximate in some way. Our assumption is that this probability of relevance can be approximated by the probability of a query given a document and relevance.

$$p(R=1|d,q) \approx p(q|d,R=1) \tag{6.9}$$

If a user likes document d, how likely would user enter query q (in order to retrieve d)? We have to be able to estimate this conditional probability without relying on the big table of relevance. But the question, now, become how can we stimate this probability? We need to use a **language models**.

6.2 Language Models

Every language models rely on the **Zipf's law**.

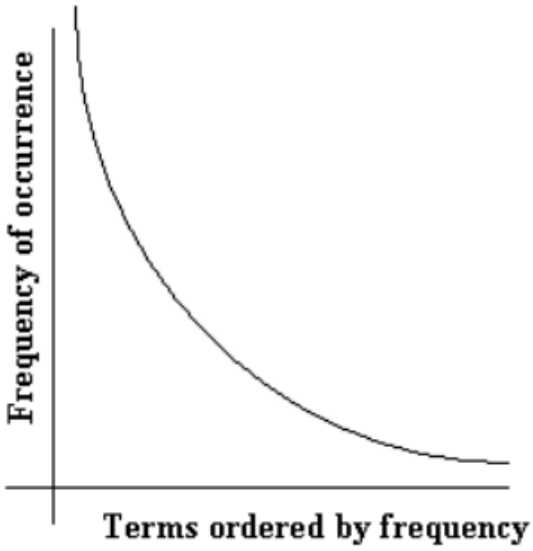
Definition 6.2.1 — Zipf's law. The distribution of word frequencies is very skewed: few words with high frequencies and many words with low frequencies.

$$r * f = k \tag{6.10}$$

The frequency of the rth most common word is inversely proportional to r.

Corollary 6.2.1 The rank of a word (r) times its frequency (f) is approximately a constant (k).

$$r * P(r) = c \tag{6.11}$$



Language models, such as unigram, bigram, and trigram models, predict the probability of a word based on its predecessors. These models are instrumental in various applications, including machine translation, spell correction, and speech recognition.
The goal of these models is to compute the probability of a sentence or sequence of words. More in detail, the goal is to compute the probability of a word w given some history h $\rightarrow P(w|h)$. How can we compute the probability? A simple solution can be estimate it from relative frequency counts; but, on large dataset, this is not feasible because there are too many possible sentences.
The first thing to do is, since the probabilities are based on counting things, counting words requires text normalization and so requires to manage punctuation, capitalizations, inflections, lemma (the canonical form of a word) and wordform (the full inflected form). After that we can compute the probability. Another option can be to use the chain rule of probability but this rule does not seem to help us, because we don't know how to compute the conditional probabilities and don't give a good approximation (only one word). But from that born the intuition of the **N-gram** model: we can approximate the history by just the last few words.

$$P(w_i|w_1 w_2 \ldots w_{i-1}) \approx P(w_i|w_{i-1}) \tag{6.12}$$

In general this is an insufficient model of language because language has long-distance dependencies but, in various application, this model gives good results.

The simples and most intuitive way to estimate probabilities is the **maximum likelihood estimation (MLE)**.

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}w_i)}{\sum_w count(w_{i-1}w)} = \frac{count(w_{i-1}w_i)}{count(w_{i-1})} \tag{6.13}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$P(\texttt{I}\,|\,\texttt{<s>}) = \frac{2}{3} = .67$ $P(\texttt{Sam}\,|\,\texttt{<s>}) = \frac{1}{3} = .33$ $P(\texttt{am}\,|\,\texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>}\,|\,\texttt{Sam}) = \frac{1}{2} = 0.5$ $P(\texttt{Sam}\,|\,\texttt{am}) = \frac{1}{2} = .5$ $P(\texttt{do}\,|\,\texttt{I}) = \frac{1}{3} = .33$

We can extend this model to N-gram with the following formula.

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{count(w_{n-N+1}^{n-1}w_n)}{count(w_{n-N+1}^{n-1})} \tag{6.14}$$

To avoid underflow, that means to have very little probabilities, we do everything in log space.

$$log(p_1 \times p_2 \times p_3 \times p_4) = logp_1 + logp_2 + logp_3 + logp_4 \tag{6.15}$$

N-grams only work well for word prediction if the test corpus looks like the training corpus. In real life, it often doesn't and so we need to train robust models that generalize. Furthermore MLE process suffers of the following problems:

- Sparse data: because any corpus is limited some acceptable word sequences are bound to be missing from it.
- **Zero-probability** N-grams that should have some non-zero probability

Smoothing Techniques

Smoothing techniques address zero-probability issues in language models. A simple tecnhique is the **Laplace Smoothing** that just add one to all the counts.

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)+1}{c(w_{i-1})+V} \tag{6.16}$$

As we can see in the figure below Laplace smoothing is a blunt instrument because add a discount factor d = $\frac{c*}{c}$ which distributes the probability over the entire column: C(want to) went from 608 to 238 and P(to|want) from .66 to .26. So it isn't used for N-grams, text classification and in domains where the number of zeros isn't so huge. A more general formulations can be the **Add-k** Smoothing.

$$P_{Add-k}(W_i \mid W_{i-1}) = \frac{c(W_{i-1},W_i)+k}{c(W_{i-1})+kV} \tag{6.17}$$

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

6.2.1 Probabilistic Retrieval Models

We quantify how likely a user would pose a particular query in order to find a particular document. We assume that the query is generated by sampling words from the document; this assumption allows us to characterize the conditional probability of a query given a document without relying on the big table that was presented earlier. This is why we can use this fundamental idea to further derive retrieval functions that we can implement with language models. We made the assumption that each query word is independent and since we are computing a query likelihood, the total probability is the probability of this particular query. The probability of the query is just a product of the probability of each query word and the probability of each word is just the relative frequency of the word in the document. We also assume that this model does not assign zero probability for any word.

$$f(q,d) = \log p(q \mid d) = \sum_{i=1}^{n} \log p(w_i \mid d) = \sum_{w \in V} c(w,q) \log p(w \mid d) \tag{6.18}$$

We can see that the basic formula of the ranking function of a probilistic language models is the same of the VSM. The key question here is what probability should be assigned to those unseen words. One idea is to let the probability of an unseen word be proportional to its probability as given by a reference language model.

$$p(w \mid d) = \begin{cases} p_{Seen}(w \mid d) & \text{if } w \text{ is seen in } d \\ \alpha_d p(w \mid C) & \text{otherwise} \end{cases} \tag{6.19}$$

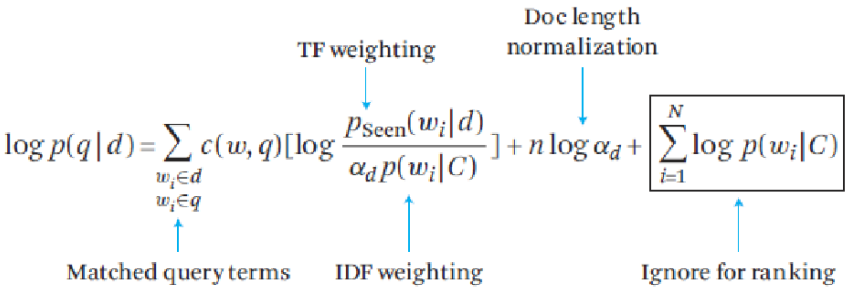
Adding this type of smoothing we can rewrite the ranking function in the following way.

$$\log p(q \mid d) = \sum_{w \in V} c(w,q) \log p(w \mid d) = \tag{6.20}$$

$$= \sum_{\substack{w \in V, \\ c(w,d)>0}} c(w,q) \log p_{Seen}(w \mid d) + \sum_{\substack{w \in V, \\ c(w,d)=0}} c(w,q) \log \alpha_d p(w \mid C) = \tag{6.21}$$

$$= \sum_{\substack{w \in V, \\ c(w,d)>0}} c(w,q) \log \frac{p_{Seen}(w \mid d)}{\alpha_d p(w \mid C)} + |q| \log \alpha_d + \sum_{w \in V} c(w,q) \log p(w \mid C) \tag{6.22}$$

We can decompose the formula used previously into two parts: the first represents the query words that appear in the document, while the second represents the query words that do not appear in the document. The latter can be further decomposed as the set of all the words in our vocabulary minus the set of words present at least once in the document. At the end we arrive to a formula very similar to the VSM. Others smoothing techniques can be the following:



- **Jelinek-Mercer smoothing**: we do a linear interpolation between the maximum likelihood estimate and the collection language model. This is controlled by the smoothing parameter $\lambda \in [0,1]$

$$p(w \mid d) = (1-\lambda)\frac{c(w,d)}{|d|} + \lambda p(w \mid C) \tag{6.23}$$

- **Dirichlet prior smoothing** or **Bayesian smoothing**

$$p(w \mid d) = \frac{c(w,d) + \mu p(w \mid C)}{|d| + \mu} = \frac{|d|}{|d| + \mu}\frac{c(w,d)}{|d|} + \frac{\mu}{|d| + \mu} p(w \mid C) \tag{6.24}$$

6.3 Search Engine Implementation

An IR system functionally consists of four components: tokenizer, indexer, ranker and feedback.

Tokenizer

Tokenizer generates the input for the system and so determines how we represent a document. The most basic tokenizer we will consider is a **whitespace tokenizer**. The process is also called feature generation because defines the building blocks of our document objects and gives us meaningful ways to compare them.

Indexer

Search engines are designed to be able to index data that is much larger than the amount of system memory; this requires us to design indexing systems that only load portions of the raw corpus in memory at one time. An **inverted index** is the main data structure used in a search engine and include two structures:

- The lexicon (a lookup table of term-specific information, such as document frequency and where in the postings file to access the per-document term counts)
- The postings file (mapping from any term integer ID to a list of document IDs and frequency information of the term in those documents).

Inverted Files are created in the following way:

1. Document are parsed one document at a time to extract tokens/terms. These are saved with the Document ID.
2. After all documents have been parsed, the inverted file is sorted alphabeticaly and in document order.

3. Multiple term/token entries for a single document are merged. Result is a triple composed of term, Document ID and term frequency.

Term	Doc #
a	2
aid	1
all	1
and	2
come	1
country	1
country	2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1
the	1
the	2
the	2
their	1
time	1
time	2
to	1
to	1
was	2
was	2

Term	Doc #	Freq
a	2	1
aid	1	1
all	1	1
and	2	1
come	1	1
country	1	1
country	2	1
dark	2	1
for	1	1
good	1	1
in	2	1
is	1	1
it	2	1
manor	2	1
men	1	1
midnight	2	1
night	2	1
now	1	1
of	1	1
past	2	1
stormy	2	1
the	1	2
the	2	2
their	1	1
time	1	1
time	2	1
to	1	2
was	2	2

If we want to score the term computer, which is term ID 56, we look up 56 in the lexicon. The information we receive could be: Term ID: 56, Document frequency: 78, Total number of occurrences: 443, Offset into postings file: 8923754. The actual lexicon would just store $56 \rightarrow \{78, 443, 8923754\}$.

Scorer

Once an inverted index is built, scoring a query term-by-term can be done efficiently on an inverted index Idx. Once we've iterated through all the query terms, the score accumulators have been finalized and we just need to sort these documents by their accumulated scores and return (usually) the top k.

6.3.1 Feedback Mechanisms

Feedback takes the results of a user's actions or previous search results to enhance the existing ranked list of results. The user can make judgements about if a returned document is useful or not. Each decision on a document is called a **relevance judgment**. Thanks to those we are able to learn what exactly is interesting to a particular user or users.

Pseudo relevance feedback

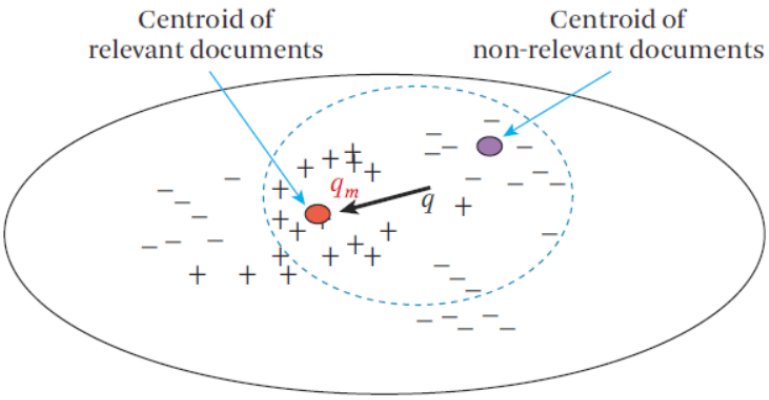
We don't have to involve users since we simply assume that the top k ranked documents are relevant. So, it's possible to learn some related terms to the query from this set anyway regardless whether the user says that a document is relevant or not. Pseudo relevance feedback is not completely reliable because we have to arbitrarily set a cutoff and we hope that the ranking function is good enough to get at least some useful documents.

Implicit feedback

In implicit feedback, we still involve users, but we don't have to explicitly ask them to make judgements. We are going to observe how the users interact with the search results by observing their clickthroughs.

Feedback in the Vector Space Model

Feedback in a TR system is based on learning from previous queries to improve retrieval accuracy in future queries. The general method in the vector space model for feedback is to modify the query vector adjusting weights of old terms or assign weights to new terms in the query vector. The query will usually have more terms, which is why this is often called **query expansion**. The most effective method for the vector space model feedback is called **Rocchio feedback**.



The query vector is in the center and the + (positive) or - (negative) represent documents. Our goal is to move the query vector to improve the retrieval accuracy, shifting the dotted circle of similarity. The formula for Rocchio feedback is:

$$\vec{q_m} = \alpha\vec{q} + \beta\frac{1}{|D_r|}\sum_{\vec{d_j}\in D_r}\vec{d_j} - \gamma\frac{1}{|D_n|}\sum_{\vec{d_j}\in D_n}\vec{d_j} \tag{6.25}$$

where $\vec{q}$ is the original query vector, $\vec{q_m}$ is the modified query vector, D_r is the set of relevant documents, D_n is the set of non-relevant documents, and α, β, γ are weights.

Imagine we have a small vocabulary,

- V = {*news, about, presidential, campaign, food, text*}

and a query

- q = {1, 1, 1, 1, 0, 0}.

		{	news	about	pres.	campaign	food	text	}
−	d_1	{	1.5	0.1	0.0	0.0	0.0	0.0	}
−	d_2	{	1.5	0.1	0.0	2.0	2.0	0.0	}
+	d_3	{	1.5	0.0	3.0	2.0	0.0	0.0	}
+	d_4	{	1.5	0.0	4.0	2.0	0.0	0.0	}
−	d_5	{	1.5	0.0	0.0	6.0	2.0	0.0	}

		{	news	about	pres.	campaign	food	text	}
+	C_r	{	$\frac{1.5+1.5}{2}$	0.0	$\frac{3.0+4.0}{2}$	$\frac{2.0+2.0}{2}$	0.0	0.0	}
−	C_n	{	$\frac{1.5+1.5+1.5}{3}$	$\frac{0.1+0.1+0.0}{3}$	0.0	$\frac{0.0+2.0+6.0}{3}$	$\frac{0.0+2.0+2.0}{3}$	0.0	}

$$\begin{aligned}\vec{q_m} &= \alpha\cdot\vec{q} + \beta\cdot C_r - \gamma\cdot C_n \\ &= \{\alpha + 1.5\beta - 1.5\gamma, \alpha - 0.067\gamma, \alpha + 3.5\beta, \alpha + 2\beta - 2.67\gamma, -1.33\gamma, 0\}.\end{aligned}$$

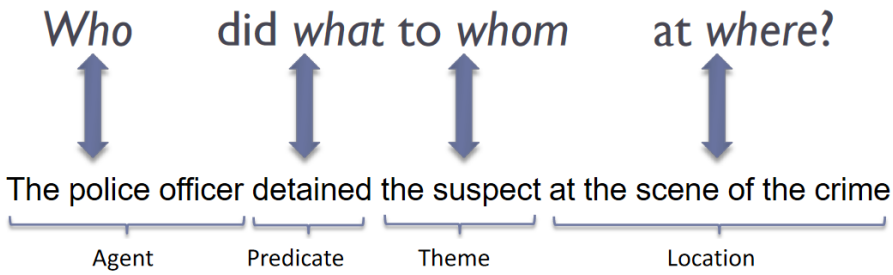
If we apply this method in practice we have to manage some potential problems:

- Negative examples or non-relevant examples tend not to be very useful and so we sometimes don't use the negative examples or set the parameter γ to be small

- It's important to avoid over-fitting, which means we have to keep relatively high weight α on the original query terms.

7. Text Classification

Text classification involves assigning subject categories, topics, or genres to a document, and it is used in various applications such as spam detection, authorship identification, age/gender identification, language identification, and sentiment analysis.



7.1 Text Classification

In terms of input and output, the process typically involves, as input, a document d and a fixed set of classes $Y = \{y_1, y_2, \ldots, y_J\}$ and, as output, a predicted class $y \in Y$. There are several methods for text classification.

1. **Hand-coded rules**: rules based on combinations of words or other features defined by experts. This method can achieve high accuracy but it is expensive to build and maintain during time because rules need to carefully refined by experts. For example, spam detection might use specific keywords like "black-list-address" or phrases such as "dollars" and "have been selected." to identify spam emails.
2. **Supervised machine learning**: model where the input includes a document d, a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$, and a training set of m hand-labeled documents $(d_1, c_1), \ldots, (d_m, c_m)$; the output is a learned classifier $\gamma : d \to c$. Various classifiers can be used in this approach, including Naïve Bayes, logistic regression, support-vector machines, and k-nearest neighbors.

Sentiment Analysis

Sentiment analysis, a subfield of text classification, is the process of analyzing digital text to determine if the emotional tone of the message is positive, negative, or neutral. It involes manage and understand the negation of positive example "I like"-"I NOT like".
To do text classification, the first step to do is compute **lexical semantics** of words in the document. It deals with any computational process involving word:

1. Word relations
   - Synonymy: words have identical or nearly identical meanings (e.g., couch/sofa),
   - Word similarity: involves relations between words based on their usage in similar contexts (vanish/disappear).
   - Word relatedness: refers to the connection between words that co-participate in shared events, such as coffee and cup.
   - Taxonomic relations include hyponymy (subclass) and hypernymy (superclass), exemplified by the relationship between car and vehicle.
2. Word sense disambiguation

7.2 Using vocabularies

How can we define the meaning of a word sense? Can we just look in a dictionary?
Dictionaries tend to use many fine-grained senses which are not understandable from a machine.

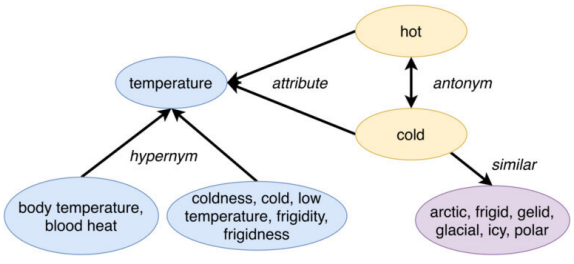
right	*adj.* located nearer the right hand esp. being on the right when facing the same direction as the observer.
left	*adj.* located nearer to this side of the body than the right.
red	*n.* the color of blood or a ruby.
blood	*n.* the red liquid that circulates in the heart, arteries and veins of animals.

Figure 7.1: The definition of right makes two direct references to itself. The entry for left contains an implicit self-reference in the phrase this side of the body, which presumably means the left side.

So for computational purposes, we can define a sense through its relationship with other senses. Sense relations of this sort are embodied in on-line databases like WordNet.

WordNet

WordNet, a lexical database for English sense relations, is hierarchically organized and includes categories like noun, verb, adjective, and adverb.



It is often used for tasks like word sense disambiguation (WSD), where the goal is to determine the correct sense of a word in context.

- **Supervised WSD**: if we have data that has been hand-labeled with correct word senses, we can use a supervised learning approach. The output of training is thus a classifier system capable of assigning sense labels to unlabeled words in context

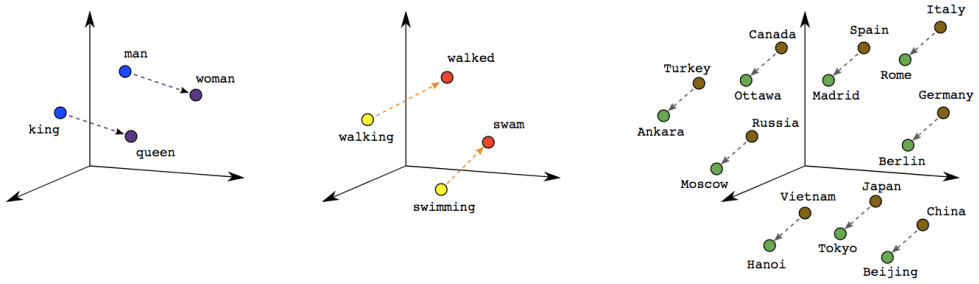
- **Dictionary Methods**: One alternative is to get indirect supervision from dictionaries and thesauruses or similar knowledge bases. The most well-studied dictionary-based algorithm for sense disambiguation is the **Lesk** algorithm that choose sense with most word overlap between gloss and context.

The **bank** can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

bank[1]	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank[2]	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

7.3 Vector Semantics and Embeddings

Word embedding or word vector is an approach with which we represent documents and words. It is defined as a numeric vector input that allows words with similar meanings to have the same representation. It can approximate meaning and represent a word in a lower dimensional space. These can be trained much faster than the hand-built models that use graph embeddings like WordNet.



Male-Female Verb Tense Country-Capital

How can we create these vectors? We have two main methods:

- Sparse vector representations
- Dense vector representations

7.3.1 Sparse vector representations

Represent words as sparse vectors and then compute similarity using a similarity function as the cosine function between two vectors (equal to the normalized dot product).

Term-document matrix

Each row represents a word in the vocabulary and each column represents a document from some collection. Each cell represents count of word w in a document d.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	0

Figure 7.2: Two documents are similar if their vectors are similar and two words are similar if their vectors are similar

Term-term matrix

Each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus.

	aardvark	...	computer	data	pinch	result	sugar	...
apricot	0	...	0	0	1	0	1	
pineapple	0	...	0	0	1	0	1	
digital	0	...	2	1	0	1	0	
information	0	...	1	6	0	4	0	

Raw word frequency is not a great measure of association between words, for example, "the" and "of" are very frequent, but maybe not the most discriminative. We'd rather have a measure that asks whether a context word is particularly informative about the target word.

- **Term frequency * inverse document frequency**
- **Positive Pointwise Mutual Information (PPMI)**: The pointwise mutual information measures of how often two events x and y occur, compared with what we would expect if they were independent

$$PMI(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)} \tag{7.1}$$

PMI values range from negative to positive infinity and negative PMI values tend to be unreliable. For this reason we take the maximum between zero and PMI value.

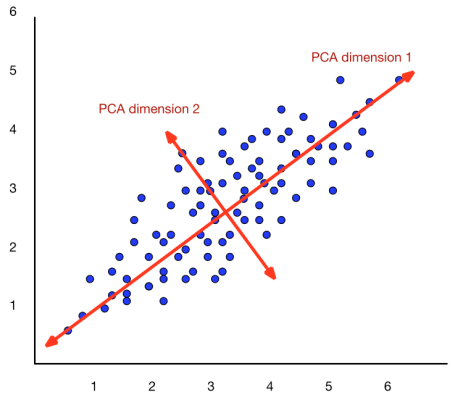
$$PPMI(w,c) = max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0) \tag{7.2}$$

Problem with sparse vectors is that have most of elements at zero (inefficient) and have a lot of weights to tune. It is best to have short vectors easier to use as features in machine learning and with less weights to tune.

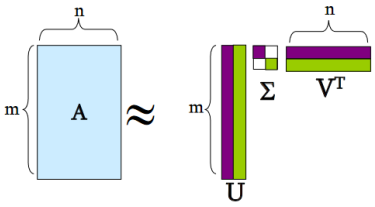
7.3.2 Dense vector representations

Singular Value Decomposition (SVD)

SVD is a method for dimensionality reduction that involves rotating axes into a new space capturing the most variance and factorizing the term-context matrix into three matrices (U, Σ, and V). It is a method for finding the most important dimensions of a data set, those dimensions along which the data varies the most. SVD is used to factorize term-context matrix M into the product of three matrices reducing the memory required. By using only the first k dimensions, of W, S, and C, the product of these 3 matrices becomes a least-squares approximation to the original M.
To improve performance we can use **truncated SVD** that use only the top k, leads to a reduced |V| x k matrix Wk, with one k-dimensioned row per word.



$$\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_i \sigma_i \mathbf{u}_i \circ \mathbf{v}_i^T$$



Latent Semantic Analysis (LSA) is an algorithm that use SVD to reduce size of sparse vectors. It can be construed simply as a practical expedient for obtaining approximate estimates of the contextual usage substitutability of words in larger text segments.

- Represent the text as a term-context matrix. Each cell contains the frequency with which the word of its row appears in the passage denoted by its column.
- Apply singular value decomposition (SVD) to the matrix.
- Reduce the dimensionality of the solution simply by deleting coefficients in the diagonal matrix (keep 80-90% of 'energy' = $\sum_i \sigma_i^2$)

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

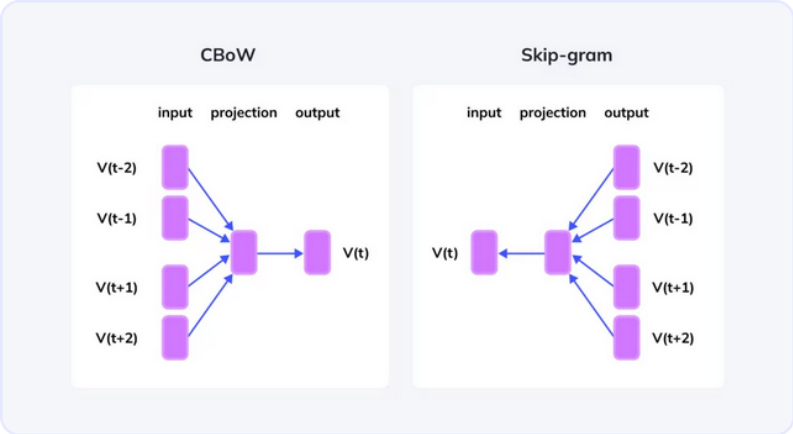
Example - LSA (5)

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
interface	0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
user	0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
system	0.45	1.23	1.05	1.27	0.56	-0.07	-0.15	-0.21	-0.05
response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
survey	0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
trees	-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
graph	-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
minors	-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

Word2Vec

Another common method is the Word2vec. The idea is, instead of counting how often each word w occurs, we train a classifier on a binary prediction task: "Is word w likely to show up near a term?" and we take the learned classifier weights as the word embeddings. Word2Vec is a two-layer neural network that processes text by taking in batches of raw textual data, processing them and producing a vector space of several hundred dimensions. Each unique word in the data is assigned a corresponding vector in the space. The positioning of these vectors in the space is determined by the words' semantic meanings and proximity to other words. The Word2Vec model can be implemented using two architectural designs: the Continuous Bag of Words (CBOW) Model and the Continuous Skip-Gram Model. Both models aim to reduce the dimensionality of the data and create dense word vectors, but they approach the problem differently

- The **Continuous Bag of Words** predicts the target word from its surrounding context words. In other words, it uses the surrounding words to predict the word in the middle. This model takes all the context words, aggregates them, and uses the resultant vector to predict the target word. For example, in the sentence "The cat sat on the mat," if we use "cat" as our target word, the CBOW model will take "The", "sat", "on", "the", "mat" as context and predict the word "cat". This model is beneficial when we have a small dataset, and it's faster than the Skip-Gram model.
- The **Continuous Skip-Gram Model**: The Skip-Gram model predicts the surrounding context words from a target word. In other words, it uses a single word to predict its surrounding context. For example, if we again take the sentence "The cat sat on the mat," the Skip-Gram model will take "cat" as the input and predict "The", "sat", "on", "the", "mat". This model works well with a large dataset and with rare words. It's computationally more expensive than the CBOW model due to its task of predicting multiple context words. However, it provides several advantages, including an improved ability to capture semantic relationships, handle rare words, and be flexible to linguistic context.



Furthermore, skip-gram model can be trained using a simple binary classifier, instead of a neural network.

- Treat the target word and a neighboring context word as positive examples.

$$P(+|w,c) \tag{7.3}$$

- Randomly sample other words in the lexicon to get negative samples

$$P(-|w,c) = 1 - P(+|w,c) \tag{7.4}$$

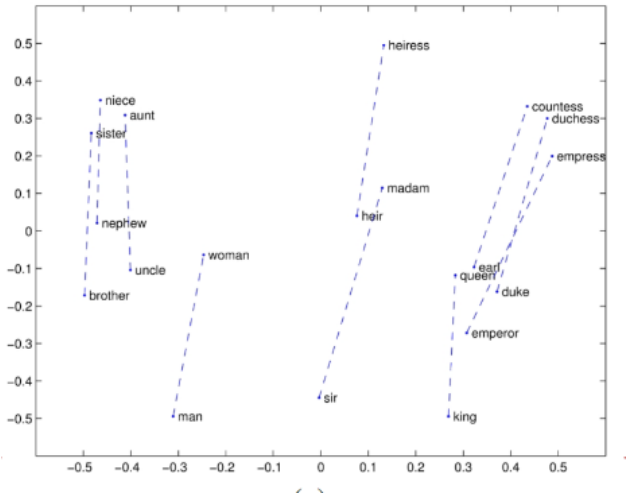
- Use logistic regression to train a classifier to distinguish those two cases
- Use the regression weights as the embeddings

To compute similarity between these dense embeddings, we rely on the intuition that two vectors are similar if they have a high dot product. Given the set of positive and negative training instances, and an initial set of embeddings, the goal of the learning algorithm is to adjust those embeddings such that we maximize the similarity of the target word, context word pairs (w,c) drawn from the positive examples and, at the same time, we minimize the similarity of the (w,c) pairs drawn from the negative examples. We can use stochastic gradient descent to train to this objective: iteratively modifying the parameters. Dot product is not a probability: it's a number ranging from 0 to inf so to turn it into a probability, we use the sigmoid function s(x). This is the probability for one word: we need to take account of the multiple context words in the window. Skip-gram makes the strong

simplifying assumption that all context words are independent, allowing us to just multiply their probabilities (sum of logaritms).

$$\log P(+ \mid w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot \mathbf{w}) \tag{7.5}$$

A semantic property of Word2Vec's embeddings is their ability to capture relational meanings, computing parallelogram model, between words. For example in the figure below, the result of the expression vector('king') - vector('man') + vector('woman') is a vector close to vector('queen').



8. Large Language Model

8.1 Neural Networks

Neural networks are computational models used for various tasks in language processing, such as classification.
The building block of a neural network is a single computational unit, that takes real numbers as input, computes a weighted sum, and applies an activation function.

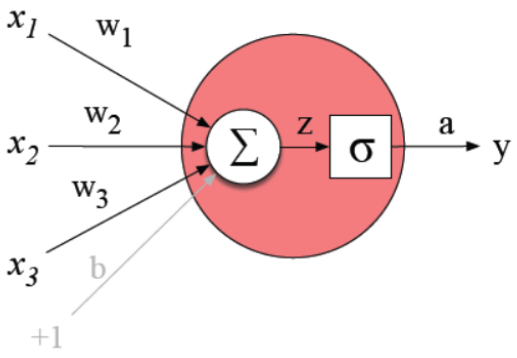
$$z = \mathbf{w}^T \mathbf{x} + b \tag{8.1}$$

where $\mathbf{w}$ is the vector of weights, $\mathbf{x}$ is the vector of inputs, and b is the bias term.
The activation functions can be the following:

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \text{(Sigmoid)} \tag{8.2}$$

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad \text{(Tanh)} \tag{8.3}$$

$$\text{ReLU}(z) = \max(0, z) \quad \text{(ReLU)} \tag{8.4}$$



8.1.1 Feed-Forward Neural Networks

A feed-forward neural network consists of multiple layers, including an input layer, hidden layers, and an output layer. Each layer is fully connected to the next, without cycles.

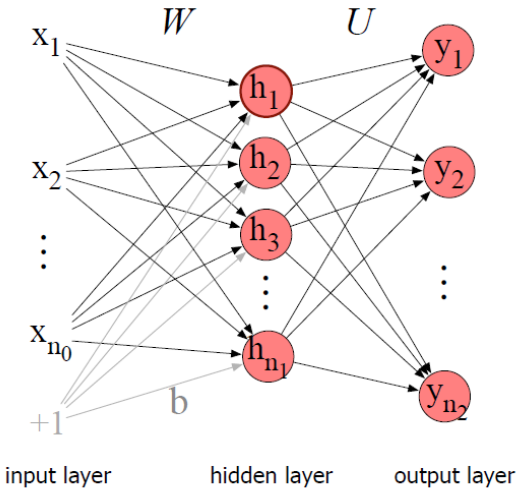


Figure 8.1: Caption

We represent the parameters for the entire hidden layer by using a single weight matrix W and a single bias vector b for the whole layer. Each element W_{ji} of the weight matrix W represents the weight of the connection from the ith input unit x_i to the jth hidden unit h_j. With this notation, the hidden layer computation for a feed-forward network can be done very efficiently with simple matrix operations.

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{8.5}$$

The output layer take this representation h and compute a final output. This output could be a real valued number, or a class in the case of classification. If we are doing a binary task, we have a single output node, and its value y is the probability of positive versus negative sentiment. Otherwise, if we are doing multinomial classification, we have one output node for each class. The output layer has a weight matrix (U).

$$\mathbf{z} = \mathbf{U}\mathbf{h} \tag{8.6}$$

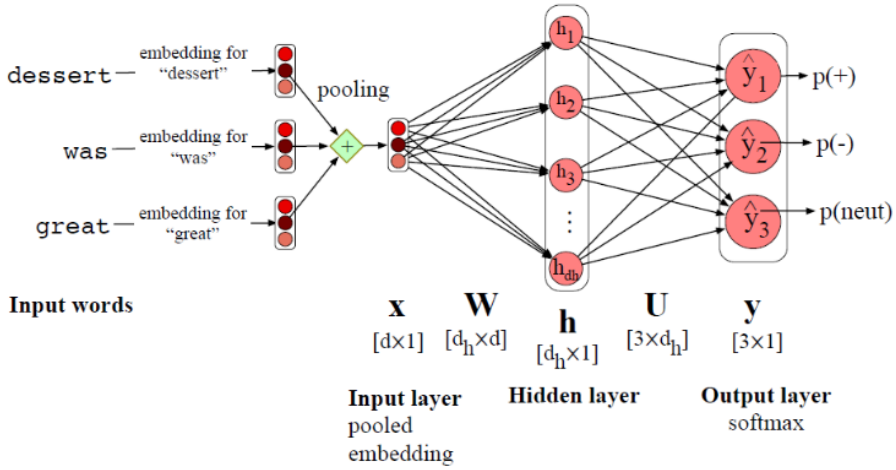
Z can't be the output of the classifier, since it's a vector of real-valued numbers; what we need for classification is a vector of probabilities. The softmax function is a convenient function for normalizing a vector of real values, by which we mean converting it to a vector that encodes a probability distribution (all the numbers lie softmax between 0 and 1 and sum to 1).

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{8.7}$$

Feed-Forward for NLP

Most applications of neural networks for NLP exploit the ability to learn features from the data by representing words as embeddings. There are various ways to represent an input for classification, the most common is apply a pooling function to the embeddings of all the words in the input.

- **Mean**: applicable because words in the same context have similar embeddings
- **Element-wise Max**



In practice we want to efficiently classify an entire test set of m examples and for do this we vectorize the process.
We pack all the input feature vectors for each input x into a single input matrix X, with each row is a row vector consisting of the pooled embedding for input example x(i). X is of shape [m x d], W is of shape [d_h x d]: we have to reorder how we multiply X and W and transpose W so they correctly multiply to yield a matrix H of shape [m x d_h]. The bias vector b has to be replicated into a matrix of shape [m x dh]. We need to similarly reorder the next step and transpose U. Finally, our output matrix Y is [m x d_o], where d_o is the number of output classes.

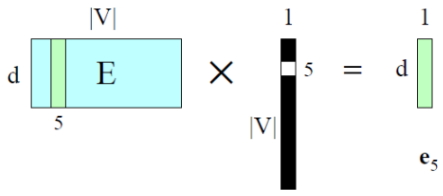
$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^T + \mathbf{b}) \tag{8.8}$$

$$\mathbf{Z} = \mathbf{H}\mathbf{U}^T \tag{8.9}$$

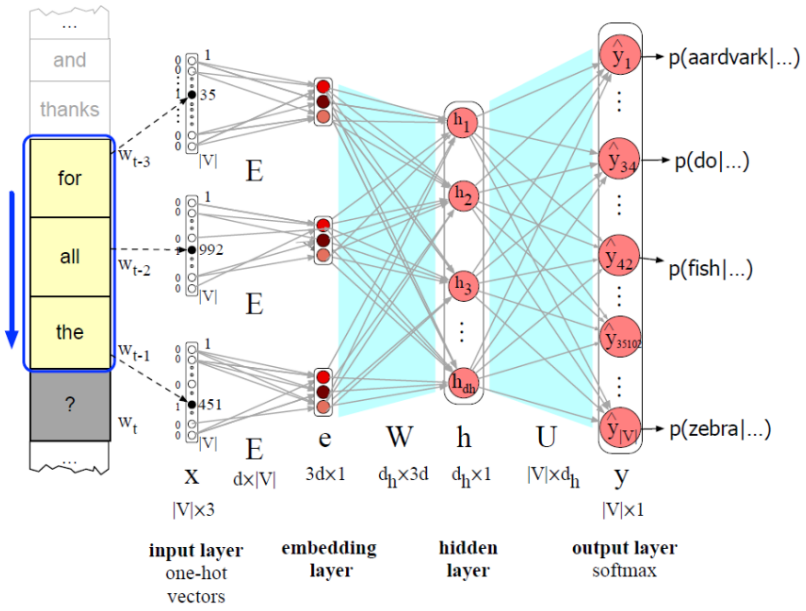
$$Y = \text{softmax}(\mathbf{Z}) \tag{8.10}$$

8.2 Neural language model

A feedforward neural language model (LM) is a feedforward network that takes as input at time t a representation of some number of previous words (wt-1;wt-2, etc.) and outputs a probability distribution over possible next words. We approximates the probability of a word given the entire prior context by means of the N -1 previous words.
Forward inference or decoding is the task, given an input, of running a forward pass on the network to produce a probability distribution over possible outputs, in this case next words. How can we produce the embeddings given N words? Each words is a one-hot vector of length vector |V| with one dimension for each word in the vocabulary. The embedding weight matrix E has a column for each word, each a column vector of d dimensions. Multiplying by a one-hot vector that has only one non-zero element $x_i = 1$ simply selects out the relevant column vector for word i, resulting in the embedding for word i.



The resulting embedding vectors are concatenated to produce the embedding layer.



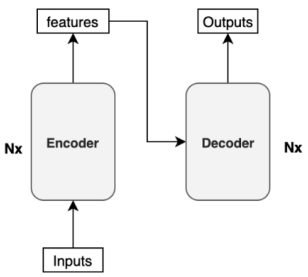
The goal of the training procedure is to learn parameters W[i] and b[i] for each layer i. We need a loss function that models the distance between the system output and the gold output: it's common to use the cross-entropy loss.

$$L = -\sum_i y_i \log(\hat{y}_i) \tag{8.11}$$

To find the parameters that minimize the loss function, we can use the gradient descent optimization algorithm, that gradient descent requires knowing the gradient of the loss function, the vector that contains the partial derivative of the loss function with respect to each of the parameters.
Neural language models based on feed-forward neural networks are a great tool to compute probabilities of possible next words with a restricted context (7/8 words). Extending the number of previous words to 15-20, the feed-forward neural networks are no longer able to compute efficiently next word.

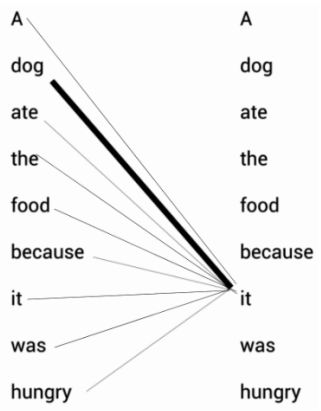
8.3 Contextualized word embeddings

The standard architecture for building large language models is the transformer, a component that makes use of a novel mechanism called self-attention. The transformer consists of an encoder-decoder architecture. We feed the input sentence (source sentence) to the encoder, that learns the representation of the input sentence and sends the representation to the decoder.The decoder receives the representation learned by the encoder as input and generates the output sentence

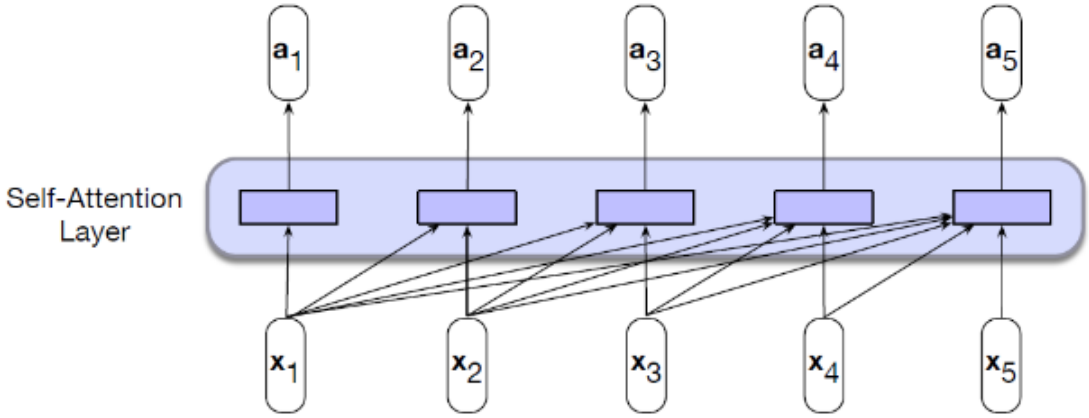


8.3.1 Self-attention mechanism

Self-attention can be thought of a way to build contextual representations of a word's meaning that integrate information from surrounding words.



A sequential model is not able to get the connection between «dog» and «it». To understand the real meaning of the sentence this connection has to be found and this can be done with a self-attention layer. A self-attention layer maps input sequences $(x)1,\ldots,x_n)$ to output sequences of the same length $(a_1,\ldots,a_n)$. When processing each item in the input, the model has access to all of the inputs up to and including the one under consideration (no access to information about inputs beyond the current one).



The computation of a3 is based on comparisons between the input x3 and its preceding elements x1 and x2, and to x3 itself. The simplest form of comparison between elements in a self-attention layer is a dot product. We normalize the scores with a softmax to create a vector of weights.

$$y_i = \sum_{j \leq i} \text{softmax}(score(x_i, x_j))x_j \tag{8.12}$$

There are three roles that each input embedding plays during the course of the attention process.

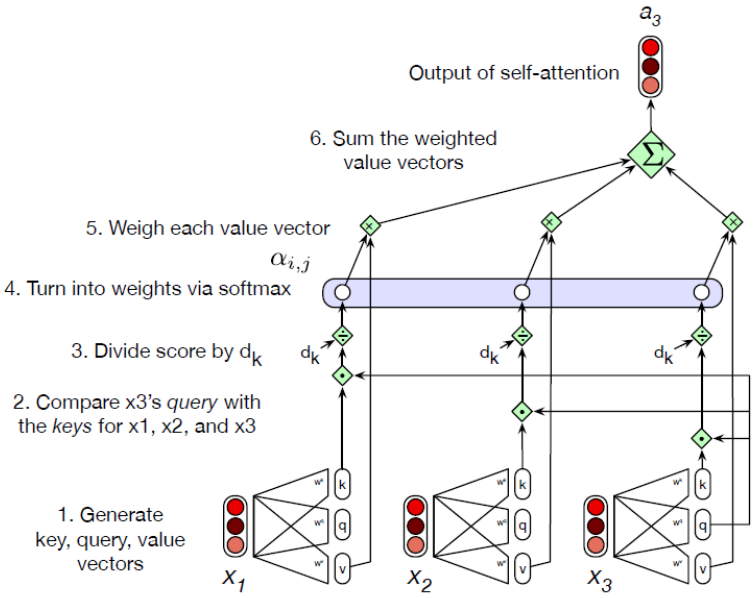
- **Query**: As the current element of attention.
- **Key**: As the previous element of attention.
- **Value**: As a value used to compute the output for the current focus of attention.

Transformers introduce weight matrices W^Q,W^K,W^V. These weights project each input vector x_i into a representation of its role as a key, query, or value.

$$q_i = W^Q x_i \tag{8.13}$$
$$k_i = W^K x_i \tag{8.14}$$
$$v_i = W^V x_i \tag{8.15}$$

The dimensions for these matrices are [d x d_k] for the key and query vectors, [d x d_v] for the value vectors. The result of a dot product can be an arbitrarily large (positive or negative) value, to avoid this, the dot product needs to be scaled.

$$\text{score}(x_i, x_j) = \frac{q_i k_j}{\sqrt{d_k}} \tag{8.16}$$



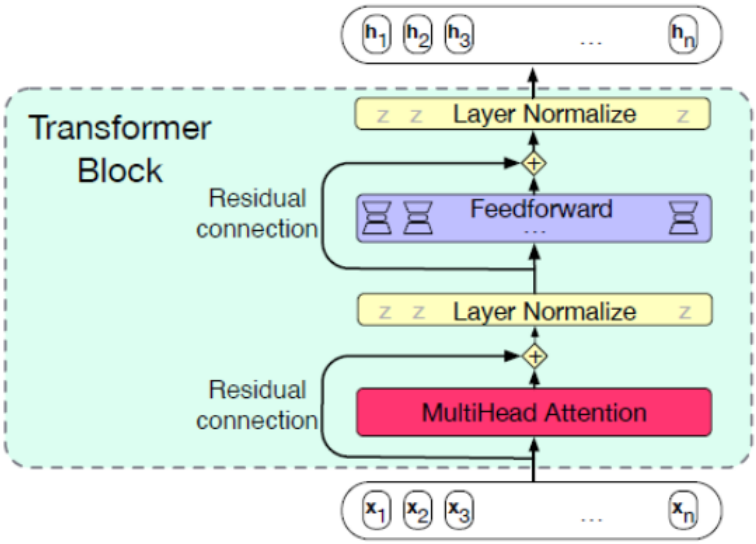
Since each output is computed independently, the entire process can be parallelized, by packing the input embeddings of the N tokens of the input sequence into a single matrix X [N x d].

$$A = \text{SelfAttention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{8.17}$$

The calculation of the comparisons in QK^T results in a score for each query value to every key value, including those that follow the query. This is inappropriate in the setting of language modeling since guessing the next word is pretty simple if you already know it. To fix this, the elements in the upper-triangular portion of the matrix are zeroed out, thus eliminating any knowledge of words that follow in the sequence.

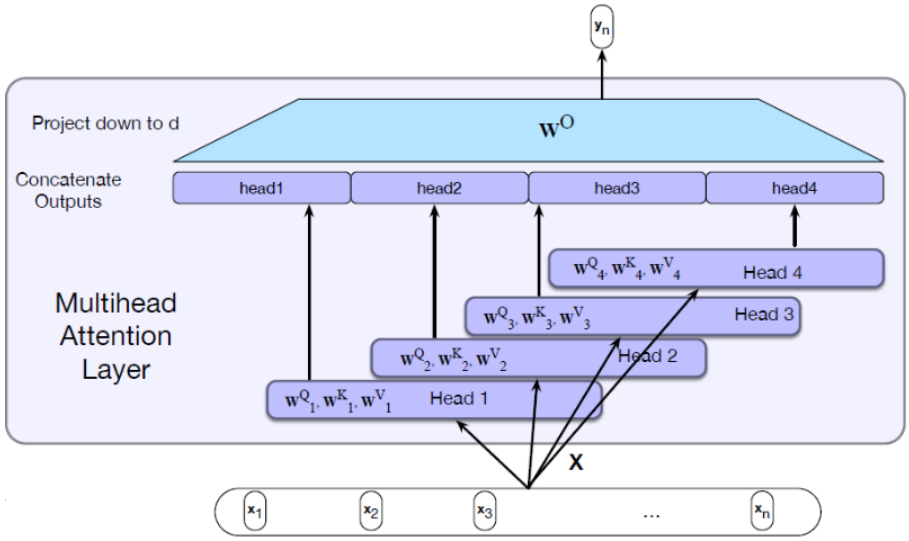
8.3.2 Transformer Block

The transformer is composed of the components shown in the figure below.



Multi-head attention mechanism

A single transformer block cannot learn to capture all of the kinds of relations among its inputs. Transformers address this issue with multi-head self-attention layers. Heads are sets of self-attention layers that reside in parallel layers at the same depth in a model, each with its own set of parameters. Each head is provided with its own set of key, query and value matrices: W_i^K , W_i^Q and W_i^V .



The output of each head is of shape [N x d_v] so the output of the multi-head layer with h heads consists of h vectors of shape is [N x d_v]. Then concatenating the outputs using a linear projection we arrive to W^O [hd_v x d]

Residual connections

Residual connections pass information from a lower layer to a higher layer without going through the intermediate layer. This improves gives higher level layers direct access to information from lower layers

Feedforward layer

The feedforward layer contains N position-wise networks, one at each position and each is a fully-connected 2-layer network. The feedforward networks are independent for each position and so can be computed in parallel. It is common that the dimensionality dff of the hidden layer of the feedforward network is larger than the model dimensionality d.

Layer Normalization

It is a form of normalization that can be used to improve training performance. It keeps the values of a hidden layer in a range that facilitates gradient-based training.

The function computed by a transformer block can be expressed:

$$O = \text{LayerNorm}(X + \text{SelfAttention}(X))H = \text{LayerNorm}(O + FFN(O)) \tag{8.18}$$

8.3.3 Positional encoding

How can we know the sequence of words? The transformer computes two embeddings: an input token embedding, and an input positional embedding. A token embedding is a vector of dimension d that provides the initial representation for the input token. These token embeddings are not position-dependent and we can combine token embeddings with positional positional embeddings specific to each position in an input sequence.
Absolute position starts with randomly initialized embeddings corresponding position to each possible input position up to some maximum length.

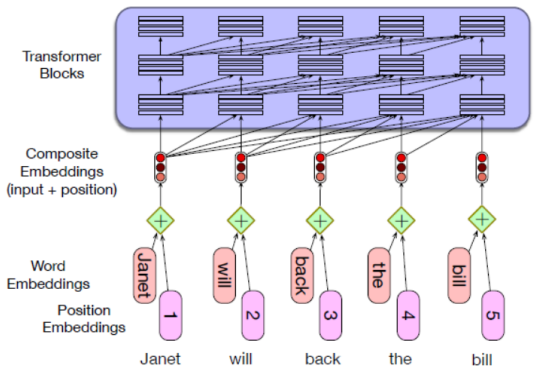
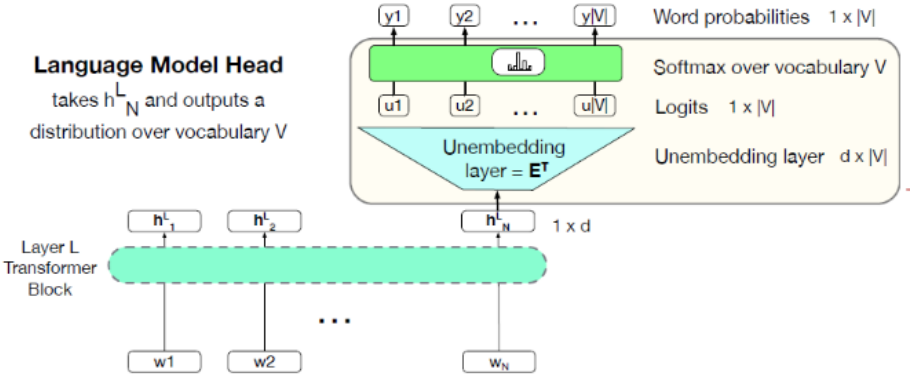


Figure 8.2: The model have to learn also the combination between positional and token embedding (possible imprecision)

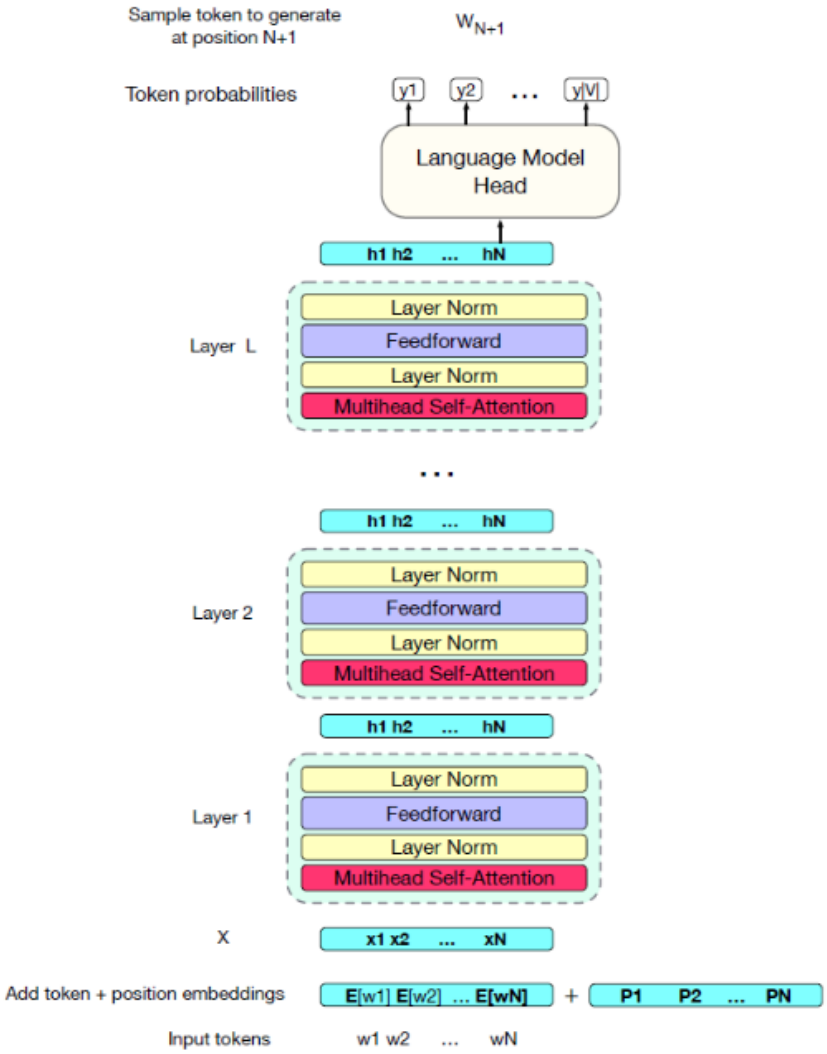
A problem with the absolute position embedding approach is that there will be plenty of training examples for the initial positions and fewer at the outer length limits. These latter embeddings may be poorly trained and may not generalize well during testing.
An alternative approach is to choose a static function that maps integer inputs to real-valued vectors in a way that captures the inherent relationships among the positions. That is, it captures the fact that position 4 in an input is more closely related to position 5 than it is to position 17. A combination of sine and cosine functions with differing frequencies was used in the original transformer work.

8.3.4 Language Modeling Head

The head is the additional neural circuitry we add on top of the basic transformer architecture to assign a probability to each possible next word.



We take the output generated by the transformer on the last sequence because it represents all our knowledge since it is based on all the sequence elements and I reverse the embedding generated on the vocabulary. A softmax layer turns the logits u into the probabilities y over the vocabulary.



Many practical NLP tasks can be cast as word prediction. We can cast sentiment analysis as language modeling by giving a language model a context, cast the question answering as word

prediction by giving a language model a question and a token like A: suggesting that an answer should come next and we cast summarization as language modeling by giving a large language model a text, and follow the text by a token like tl;dr('too long; don't read').
We can do that with different approaches:

- **Greedy decoding**: In greedy decoding, at each time step in generation, the output y_t is chosen by computing the probability for each possible outputs (every word in the vocabulary) and then choosing the highest probability word. A major problem is that because the words it chooses are (by definition) extremely predictable, the resulting text is generic and often quite repetitive.
- **Random sampling**: Sampling from a model's distribution over words means to choose random words according to their probability assigned by the model. At each step we'll sample words according to their probability conditioned on our previous choices, and we'll use a transformer language model as the probability model that tells us this probability.
- **Top-k sampling**: It is a simple generalization of greedy decoding. We first truncate the distribution to the top k most likely words, renormalize to produce a legitimate probability distribution, and then randomly sample from within these k words according to their renormalized probabilities.
- **Nucleus or top-p sampling**: top-p sampling or nucleus sampling keep the top p percent of the probability mass. The goal is the same: to truncate the distribution to remove the very unlikely word, but by measuring probability rather than the number of words, the hope is that the measure will be more robust in very different contexts, dynamically increasing and decreasing the pool of word candidates
- **Temperature sampling**: we reshape the distribution to change probability of probable words and rare words. We implement this by simply dividing the logit by a temperature parameter t before we normalize it by passing it through the softmax.

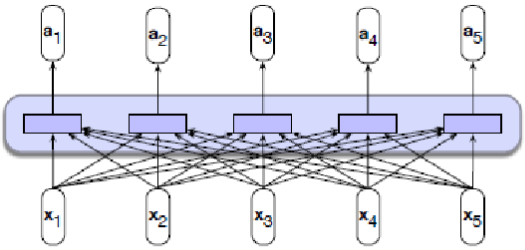
$$y = \text{softmax}(u/t) \tag{8.19}$$

When t is close to 1 the distribution doesn't change much; the lower t<1 is, the larger the scores being passed to the softmax increasing probabilities of the most high-probability words and decreasing probabilities of the low probability word. Finally when t>1 we flatten the word probability distribution.

8.4 BERT

BERT(Bidirectional Encoder Representations from Transformers) is a transformer-based model that generates context-based word embeddings. BERT is bidirectional, because the encoder of the transformer is bidirectional in nature since it can read a sentence in both directions. For example, we have the sentence A: 'He got bit by Python'. We feed this sentence as an input to the transformer's encoder and get the contextual representation (embedding) of each word in the sentence as an output. Once we feed the sentence as an input to the encoder, the decoder understands the context of each word in the sentence using the multi-head attention mechanism and returns the contextual representation of each word in the sentence as an output.
The researchers of BERT have presented the model in two standard configurations:

- **BERT-base**: it consists of 12 encoder layers, each stacked one on top of the other with all the encoders use 12 attention heads. The feedforward network in the encoder consists of 768 hidden units.
- **BERT-large**: it consists of L=24 encoder layers, each stacked one on top of the other with all the encoders use A=16 attention heads.



b) A bidirectional self-attention layer

8.4.1 Pre-training the BERT model

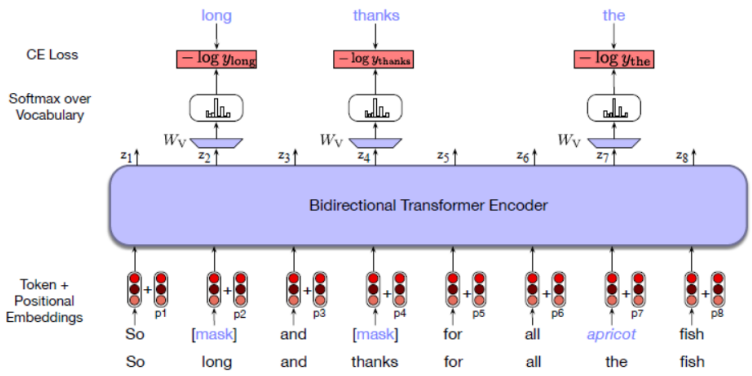
Since the model is already trained on a huge dataset, instead of training a new model from scratch for a new task, we use the pre-trained model, and adjust (fine-tune) its weights according to the new task.The BERT model is pre-trained on a huge corpus using two tasks:

- Masked language modeling
- Next sentence prediction

Masked Language Model

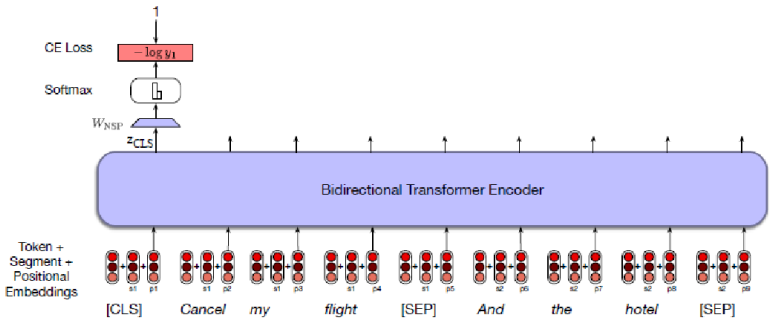
Certain words and tokens in a specific input are randomly masked or hidden in this approach and the model is then trained to predict these masked elements by using the context provided by the surrounding words. Token is used in one of three ways:

- It is replaced with the unique vocabulary token [MASK].
- It is replaced with another token from the vocabulary, randomly sampled based on token unigram probabilities.
- It is left unchanged



Next Sentence prediction

Where MLM teaches BERT to understand relationships between words — NSP teaches BERT to understand longer-term dependencies across sentences. The model is presented with pairs of sentences and predicts whether each pair consists of a pair of adjacent sentences from the training corpus. A new [CLS] token is inserted at the beginning of the first sentence and a new [SEP] token at the end of every sentence. During training, the output vector from the final layer associated with the [CLS] token represents the next sentence prediction.

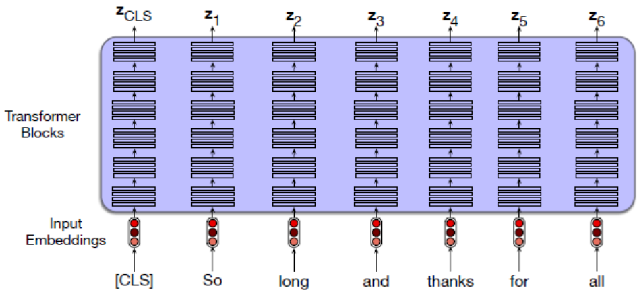


To perform classification, we take the representation of the [CLS] token, feed it to the feedforward network with the softmax function and returns the probability of our sentence pair being isNext and notNext. The [CLS] token basically holds the aggregate representation of all the tokens.

After training the model how can we use BERT to create embeddings?
Given a pretrained language model and a novel input sentence, the contextual embeddings for each token in embeddings the input is the contextual sequence of model outputs. These embeddings are vectors representing some aspect of the meaning of a token in context.

- We use the output vector z_i from the final layer of the model as a representation of the meaning of token x_i.
- Compute a representation for x_i by averaging the output tokens z_i from each of the last four layers of the model.

We use contextual embeddings as representations of word meanings in context for any task that might require a model of word meaning.



In addition, BERT uses a special type of tokenizer called a WordPiece tokenizer. If the word is present in a vocabulary, then it uses it as a token; instead, if the word is not present in the vocabulary, then it splits the word into subwords and checks whether the subword is present in the vocabulary.

In this way, we keep splitting and check the subword with the vocabulary until we reach individual characters.

The power of pretrained language models lies in their ability to extract generalizations from large amounts of text—generalizations that are useful for myriad downstream applications. The fine-tuning process consists of using labeled data about the application to train these additional application-specific parameters.

- **Sequence Classification**.
- **Pair-Wise Sequence Classification**: paraphrase detection and logical entailment.

III Tools&Practice for AI

9. Interpretability

Why do we not just trust the model and ignore why it made a certain decision? “The problem is that a single metric, such as classification accuracy, is an incomplete description of most real-world tasks". Interpretability refers to the degree to which a human can understand the cause of a decision and consistently predict the model’s result.The key benefits include:

- **Fairness**: Ensuring that predictions are unbiased and do not implicitly or explicitly discriminate against underrepresented groups.
- **Privacy**: Ensuring that sensitive information in the data is protected.
- **Robustness**: Ensuring that small changes in the input do not lead to large changes in the prediction.
- **Causality**: Check that only causal relationships are picked up.
- **Trust**: It is easier for humans to trust a system that explains its decisions compared to a black box.

9.1 Interpretability Methods

Interpretability methods are divided in two main categories:

- **Intrinsic Methods**: ML models that are considered interpretable due to their simple structure, such as short decision trees or sparse linear models
- **Post hoc Methods**: Applied after the model has been trained to interpret its predictions.

Interpretability methods are able to provides summary statistics for each feature, such as feature importance or pairwise feature interaction strengths. Furthermore they focus on the internal parameters of intrinsically interpretable models, such as weights in linear models or decision tree structures and involve methods that return or generate specific data points to make a model interpretable. Finally, can also interpret black box models by approximating them with simpler, interpretable models, either globally or locally. The simpler model is then analyzed using its internal parameters or feature summary statistics. Interpretability methods can also be divide on other factors.

Specific vs Agnostic

- **Model-specific**: Tailored to specific model types. Examples include tree-specific visualization for decision trees and coefficient interpretation for linear models.
- **Model-agnostic**: Can be applied to any model type after training. Examples include LIME and SHAP.

Local vs Global

- **Local Methods**: Explain individual predictions. Focus on a single instance to understand why a specific prediction was made. Locally, the prediction might only depend linearly or monotonically on some features, rather than having a complex dependence on them. In general, local explanations can be more accurate than global explanations. Examples include Local Surrogate Models like LIME.
- **Global Methods**: Explain the overall model behavior. Focus on the entire model to understand general patterns and feature importance. Examples include global feature importance and PDP.

Key properties for effective explanations include:

- **Accuracy**: How well the explanation predicts unseen data.
- **Fidelity**: How closely the explanation approximates the model's predictions.
- **Consistency**: Consistency of explanations across different models trained on the same task.
- **Stability**: Similarity of explanations for similar instances.
- **Comprehensibility**: Ease with which humans understand the explanations.
- **Degree of Importance**: Reflects the importance of features or parts of the explanation.
- **Representativeness**: Coverage of the explanation over multiple instances.

9.2 Interpretable Models

9.2.1 Linear Regression

Predicts the target as a weighted sum of feature inputs, with an errors that follow a Gaussian distribution.

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \varepsilon \tag{9.1}$$

Linearity makes the estimation procedure simple and easy to understand.

- Numerical features: Change in feature value directly affects the outcome by its weight.
- Binary features: Change from reference category alters the outcome by the feature's weight.
- Categorical features: Depends on encoding (one-hot or numerical).

The importance of a feature in a linear regression model can be measured by the absolute value of its **t-statistic**. The importance of a feature increases with increasing weight.

$$t_{\hat{\beta}_j} = \frac{\hat{\beta}_j}{SE(\hat{\beta}_j)} \tag{9.2}$$

Features are usually measured on different scales so the estimated weights is more comparable if we scaling the features (zero mean and standard deviation of one) before fitting the linear model. The weights of the linear regression model can be more meaningfully analyzed when they are multiplied by the actual feature values. The effect plot can help you understand how much the combination of weight and feature contributes to the predictions in your data.

$$\text{effect}_j^{(i)} = w_j x_j^{(i)} \tag{9.3}$$

The categorical feature effects can be summarized in a single boxplot, compared to the weight plot, where each category has its own row.

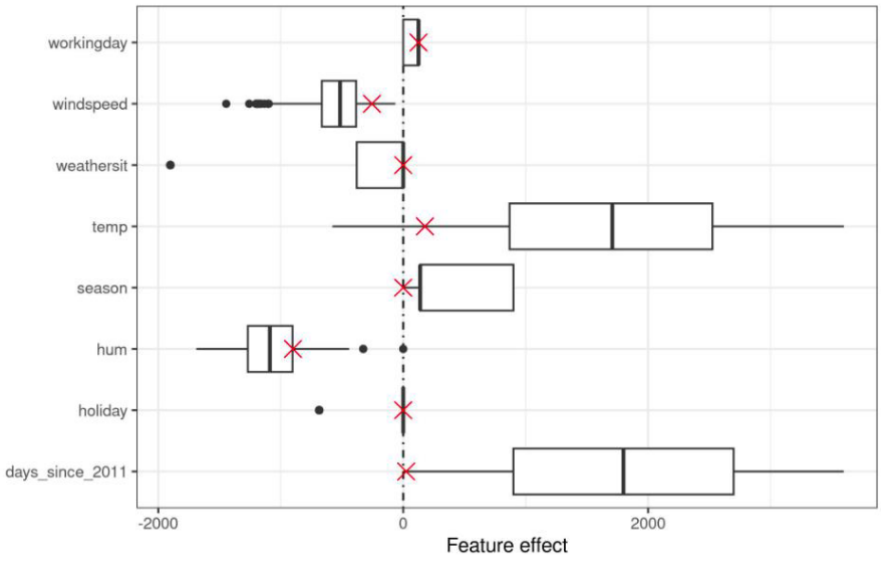


Figure 9.1: Computing max, min and median we can visualize the effect of each feature on the actual prediction

9.2.2 Logistic Regression

The interpretation of the weights in logistic regression differs from linear regression, since the outcome in logistic regression is a probability between 0 and 1.

$$\ln\left(\frac{P(y=1)}{1-P(y=1)}\right) = \log\left(\frac{P(y=1)}{P(y=0)}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p \tag{9.4}$$

$$\frac{P(y=1)}{1-P(y=1)} = \text{odds} = \exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p\right) \tag{9.5}$$

$$\frac{\text{odds}_{x_j+1}}{\text{odds}_{x_j}} = \frac{\exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_j(x_j+1) + \ldots + \beta_p x_p\right)}{\exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_j x_j + \ldots + \beta_p x_p\right)} = \exp\left(\beta_j(x_j+1) - \beta_j x_j\right) = \exp(\beta_j) \tag{9.6}$$

A change in a feature by one unit changes the odds ratio (multiplicative) by a factor of $\exp(\beta_j)$. A change in x_j by one unit increases the log odds ratio by the value of the corresponding weight. For example, if you have odds of 2, it means that the probability for y=1 is twice as high as y=0. These are the interpretations for the logistic regression:

- Numerical features: Change affects the odds ratio by a factor of exp(weight).
- Binary categorical features: Change affects odds ratio similar to numerical feature.
- Multiple categories: One solution to deal with multiple categories is one-hot-encoding, meaning that each category has its own column.

For example, as shown in the figure below, an increase in the number of diagnosed STDs (sexually transmitted diseases) changes (increases) the odds of cancer vs. no cancer by a factor of 2.27, when all other features remain the same. For women using hormonal contraceptives, the odds for cancer vs. no cancer are by a factor of 0.89 lower, compared to women without hormonal contraceptives, given all other features stay the same.

	Weight	Odds ratio	Std. Error
Intercept	-2.91	0.05	0.32
Hormonal contraceptives y/n	-0.12	0.89	0.30
Smokes y/n	0.26	1.30	0.37
Num. of pregnancies	0.04	1.04	0.10
Num. of diagnosed STDs	0.82	2.27	0.33
Intrauterine device y/n	0.62	1.86	0.40

9.2.3 Decision Trees

Tree based models split the data multiple times according to certain cutoff values in the features. After the best cutoff per feature has been determined, the algorithm selects the feature for splitting that would result in the best partition in terms of the variance. To predict the outcome in each leaf node, the average outcome of the training data in this node is used. The interpretation is simple: starting from the root node, you go to the next nodes and the edges tell you which subsets you are looking at. Once you reach the leaf node, the node tells you the predicted outcome. All the edges are connected by 'AND'. The overall importance of a feature is computed going through all the splits for which the feature was used and measure how much it has reduced the variance.

9.3 Model-Agnostic Methods

The great advantage of model-agnostic interpretation methods over model-specific ones is their flexibility:

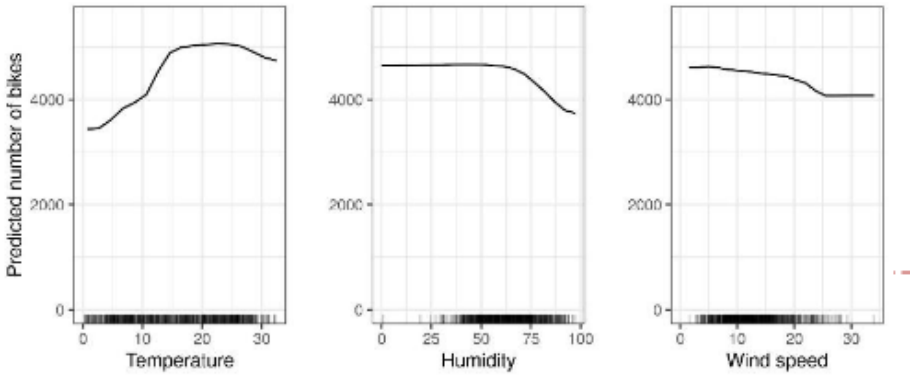
- **Model flexibility**: The interpretation method can work with any ML model.
- **Explanation flexibility**: You are not limited to a certain form of explanation. In some cases it might be useful to have a linear formula, in other cases a graphic with feature importances.
- **Representation flexibility**: The explanation system should be able to use a different feature representation as the model being explained.

9.3.1 Partial Dependence Plot (PDP)

The partial dependence plot (short PDP or PD plot) shows the marginal effect one or two features have on the predicted outcome of a ML mode.

$$\hat{f}_S(x_S) = \frac{1}{n}\sum_{i=1}^{n}\hat{f}(x_S, x_C^{(i)}) \tag{9.7}$$

The x_S are the features for which the partial dependence function should be plotted and x_C are the other features used in the machine learning model f, which are here treated as random variables. The partial function tells us for given value(s) of features S what the average marginal effect on the prediction is. In this formula, $x^{(i)}C$ are actual feature values from the dataset for the features in which we are not interested, and n is the number of instances in the dataset. An assumption of the PDP is that the features in C are not correlated with the features in S.



9.3.2 Permutation Feature Importance (PFI)

PFI measures the increase in the prediction error of the model after we permuted the feature's value. We measure the importance of a feature by calculating the increase in the model's prediction error after permuting the feature. A feature is "important" if shuffling its values increases the model error; instead, a feature is "unimportant" if shuffling its values leaves the model error. Estimate the original model error $e_{orig} = L(y, f(X))$. Then for each feature $j \in \{1, \ldots, p\}$ do the following steps:

- Generate feature matrix X_{perm} by permuting feature j in the data X. This breaks the association between feature j and true outcome y.
- Estimate error $e_perm = L(Y, f(X_perm))$ based on the predictions of the permuted data
- Calculate permutation feature importance as quotient $FI_j = e_{perm}/e_{orig}$ or difference $FI_j = e_{perm} - e_{orig}$

Always computing feature importance on test data than on training data. The reason, shown in the figure below, is that feature importance based on the training data shows many important features (overfit), while computed on unseen test data, the feature importances are close to a ratio of one (unimportant).

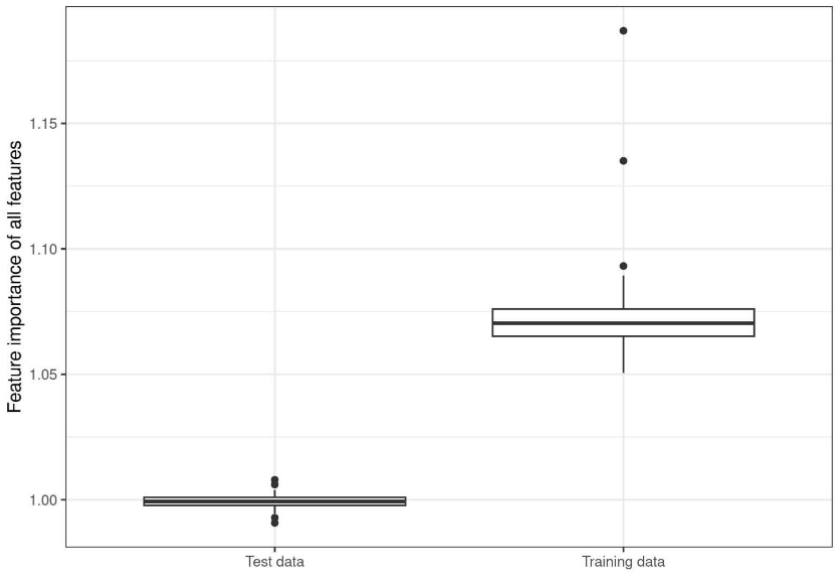


Figure 9.2: The feature importance based on training data makes us mistakenly believe that features are important for the predictions, when in reality the model was just overfitting and the features were not important at all.

9.3.3 Leave-One-Covariate-Out (LOCO)

Similar to PFI but involves removing one feature and retraining the model to measure its impact on performance.
Estimate the original model error $e_orig = L(y, f(X))$. Then for each feature $j \in \{1, \ldots, p\}$do the following steps:

- Generate feature matrix X_{-j} by removing feature j in the data X.
- Refit model f_{-j} with data X_{-j}
- Estimate error $e_{-j} = L(Y, f_{-j}(X_{-j}))$ based on the predictions of the reduced data
- Calculate LOCO Feature importance as quotient $FI_j = e_{-j}/e_{orig}$ or difference $FI_j = e_{-j} - e_{orig}$

9.4 Local Model-Agnostic Methods

Local interpretation methods explain individual predictions. Focus on a single instance to understand why a specific prediction was made.

9.4.1 Local Surrogate Models (LIME)

Local surrogate models are interpretable models used to explain individual predictions of black box ML models.Instead of training a global surrogate model, LIME focuses on training local surrogate models to explain individual predictions. LIME tests what happens to the predictions with variations of your data into the machine learning model, generating a new dataset consisting of perturbed samples and the corresponding predictions of the black box model. On this dataset LIME trains an interpretable model, which is weighted by the proximity of the sampled instances to the instance of interest. The learned model has a good approximation of the ML model predictions locally, but it does not have a good global approximation.

$$\text{explanation}(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g) \tag{9.8}$$

The explanation model for x is the model g part of the family of possible explanations G (all possible linear regression models) that minimizes loss L, which measures how close the explanation is to the prediction of the original model f, while the model complexity $\omega(g)$ is kept low.

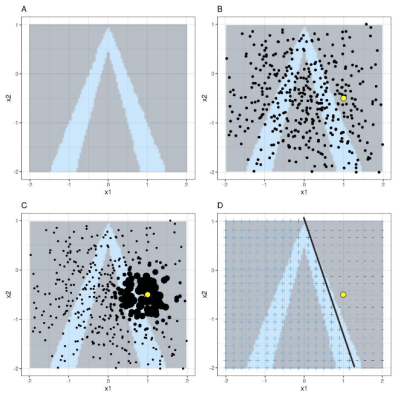


Figure 9.3: **A)** Random forest predictions given features x1 and x2. Predicted classes: 1 (dark) or 0 (light). **B)** Instance of interest (big dot) and data sampled from a normal distribution (small dots). **C)** Assign higher weight to points near the instance of interest. **D)** Signs of the grid show the Classifications of the locally learned model from the weighted samples. The white line marks the decision boundary

To assign weights to the features we can use an exponential kernel defined on some distance function D.

$$\pi x(z) = \exp(-D(x,z)^2/\sigma^2) \tag{9.9}$$

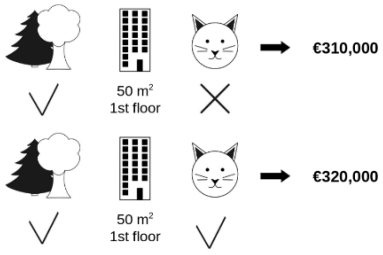
9.4.2 Counterfactual Explanations

A counterfactual explanation describes a causal situation in the form: "If X had not occurred, Y would not have occurred". In interpretable machine learning, counterfactual explanations can be used to explain predictions of individual instances; the relationship between the inputs and the prediction is very simple: the feature values cause the prediction. A counterfactual explanation of a prediction describes the smallest change to the feature values that changes the prediction to a predefined output.
For example, Peter applies for a loan and gets rejected by the banking software; the question of "why his application was rejected" can be formulated as a counterfactual: What is the smallest change to the feature that would change the prediction from rejected to approved? One possible answer could be: If Peter would earn 10,000 more per year, he would get the loan.
In another example, Anna wants to rent out her apartment, but she is not sure how much to charge for it, so she decides to train a machine learning model to predict the rent. By tweaking only the feature values under her control, she finds out that if she allows pets and installs windows with better insulation, she can charge 1000 EUR.
Counterfactuals suffer from the **'Rashomon effect'**. Each counterfactual tells a different "story" of how a certain outcome was reached. One counterfactual might say to change feature A, the other counterfactual might say to leave A the same but change feature B, which is a contradiction. This issue of multiple truths can be addressed either by reporting all counterfactual explanations or by having a criterion to evaluate counterfactuals and select the best one. How do we define a good counterfactual explanation?

- A counterfactual instance should produce the predefined prediction as closely as possible.
- A counterfactual should be as similar as possible to the instance regarding feature values.
- The counterfactual should also change as few features as possible.
- It is desirable to generate multiple diverse counterfactual explanations so that the decision-subject gets access to multiple viable ways of generating a different outcome.

9.4.3 Shapley Values

To explain the prediction we assume that each feature value of the instance is a "player" in a game where the prediction is the payout. For example, you have trained a ML model to predict apartment prices. For an apartment it predicts €300,000: you need to explain the prediction. The apartment has certain features (an area of 50 m2, is located on the 2nd floor, has a park nearby and cats are banned). The average prediction for all apartments is €310,000. How much has each feature value contributed to the prediction compared to the average prediction? Our goal is to explain the difference (-€10,000) between the actual (€300,000) and the average prediction (€310,000). The Shapley value is the average marginal contribution of a feature value across all possible coalitions.



The figure shows the contribution of cat-banned when it is added to a coalition of park-nearby and area-50. The contribution of cat-banned was €310,000 - €320,000 = -€10,000. This estimate depends on the values of the randomly drawn apartment. We will get better estimates if we repeat this sampling step and average the contributions. We repeat this computation for all possible coalitions and at the end the Shapley value will be the (weighted) average of all the marginal contributions to all possible coalitions.

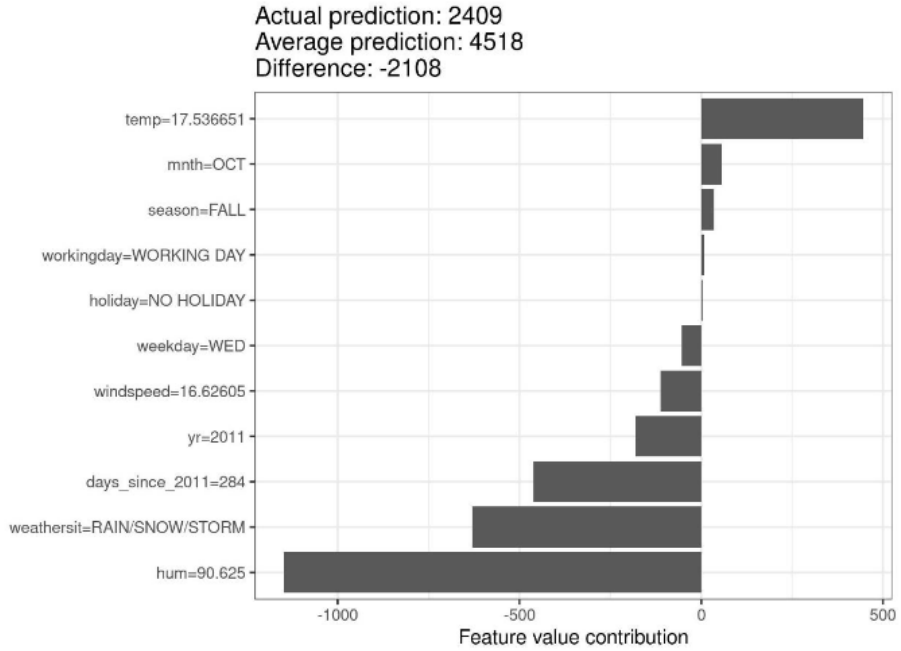


Figure 9.4: Shapley values for day 285. This day is -2108 below the average prediction of 4518. The weather situation and humidity had the largest negative contributions. The temperature on this day had a positive contribution. The sum of Shapley values yields the difference of actual and average prediction (-2108).

N is the set containing the players. S is a subset of N and i is an element of N. V is a value function that maps subsets of players S to a real number v(S). When a player i join a set of players S, the marginal contribution of player i to S is:

$$V(S \cup i) - v(S) \tag{9.10}$$

The marginal contribution measures the value that player i added when he joined the group of players S. The Shapley value of player i given the set N and the value function v is defined in the following way.

$$\phi_i(N,v) = \underbrace{\frac{1}{|N|!}\sum_{S \subseteq N \setminus \{i\}}}_{\text{Average}} \overbrace{|S|!\,(|N|-|S|-1)!}^{\text{Weight}} \overbrace{[v(S \cup \{i\}) - v(S)]}^{\text{Marginal contributions}}$$

Approximate Shapley estimation

The computation time increases exponentially with the number of features.
One solution to keep the computation time manageable is to compute contributions for only a few samples of the possible coalitions.
Another is to fix a number of iterations M and for all m = $1,\ldots,M$ doing the following step:

- Choose a random permutation o of the feature values
- Construct two new instances (adding random value z)
  - One with j: $x{+}j = (x_{(1)},\ldots,x_{(j-1)},x_{(j)},z_{(j+1)},\ldots,z_{(p)})$
  - One without j: $x_{-j} = (x_{(1)},\ldots,x_{(j-1)},z_{(j)},z_{(j+1)},\ldots,z_{(p)})$
- Compute marginal contribution: $\phi_{mj} = f(x_{+j}) - f(x_{-j})$ to understand the impact of feature j.

At the end compute Shapley value as the average.

$$\phi_j(x) = \frac{1}{M}\sum_m \phi_{mj} \tag{9.11}$$

The Shapley value is the only explanation method with a solid theory. The axioms, **efficiency, symmetry, dummy, additivity**, give the explanation a reasonable foundation.

- **Efficiency**: All the revenues v(N) of the grand coalition N are redistributed among all the players (no more, no less)
- **Symmetry**: Players i and j who contribute the same to all coalition subsets S receive the same share
- **Linearity**: Let (N, v_1) and (N, v_2) be two coalition games, the values from the games can be combined in an additive way.
- **Dummy player**: Those who do not contribute receive nothing

The Shapley value allows contrastive explanations. Instead of comparing a prediction to the average prediction of the entire dataset, you could compare it to a subset or even to a single data point. This contrastiveness is something that local models like LIME do not have.
Shapley value has also a lot of disadvantages:

- The Shapley value requires a lot of computing time
- The Shapley value can be misinterpreted
- The Shapley value is the wrong explanation method if you seek sparse explanations (explanations that contain few features) because explanations always use all the features.
- The Shapley value returns a value per feature, but no prediction model like LIME.
- Like many other permutation-based interpretation methods, the Shapley value method suffers from inclusion of unrealistic data instances when features are correlated.

9.4.4 SHAP (SHapley Additive exPlanations)

SHAP (SHapley Additive exPlanations) is a method to explain individual predictions based on Shapley values. The goal of SHAP is to explain the prediction of an instance x by computing the contribution of each feature to the prediction. One innovation that SHAP brings to the table is that the Shapley value explanation is represented as an additive feature attribution method, a linear model.

$$g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j z_j' \tag{9.12}$$

g is the explanation model,$z' \in \{-,1\}$ M is the coalition vector and the maximum coalition size, ϕ_0 is the null output (the average value) and ϕ_j feature effect.

KernelSHAP

KernelSHAP, an alternative, kernel-based estimation approach for Shapley values inspired by local surrogate models:

- Create a new dataset sampling coalitions $z_k^{'}$
- Compute the weight for each $z_k^{'}$ with the SHAP kernel
- Fit weighted linear model
- Return Shapley values ϕ_k, the coefficients from the linear model

TreeSHAP

TreeSHAP, an efficient estimation approach for tree-based models.

10. Fairness

Machine learning (ML) can significantly enhance decision-making by uncovering complex and subtle relationships in historical data that humans might miss. However, ML's reliance on generalizing from examples through induction brings inherent risks. The evidence-based nature of ML doesn't inherently guarantee accuracy, reliability, or fairness. The quality and bias in the historical data used for training can lead to models that replicate existing societal biases and inequalities.
Disparities observed in ML outcomes do not necessarily indicate intentional discrimination. It's crucial to determine if these disparities are justified and whether they cause harm. For instance, Amazon's delivery system showed racial disparities, with White residents being more likely to receive same-day delivery than Black residents. Although race wasn't an explicit factor, the system inadvertently perpetuated racial inequalities.

10.1 Bias

The term bias is often used to refer to demographic disparities in algorithmic systems that are objectionable for societal reasons. Friedman and Nissenbaum (1996) identified three types of bias that can arise in computer systems:

- preexisting
- technical
- emergent

Preexisting Bias

Originates from societal biases reflected in input data. For this reason, the presence or absence of this type of bias cannot be scientifically verified. Requires understanding and adjustments based on belief systems about the bias's nature. Detecting and mitigating preexisting bias is the subject of much research under the heading of algorithmic fairness.
For example, consider the use of an applicant's Scholastic Assessment Test (SAT) score during the screening stage. It has been documented that it differs across racial groups. But the score itself says more about socioeconomic conditions than an individual's academic potential. So we may then seek to correct for that bias before using the feature.

(a) Data is an image of the world, its mirror reflection. When we think about bias in the data, we interrogate this reflection. Does the data systematically over-represent or under-represent some parts of the world, or does it distort reality in some other way?

(b) Even if we were able to reflect the world perfectly in the data, it would still be a reflection of the world as it is, not how it could or should be. People—not data or algorithms—must decide what world we want to live in.

(c) Debiasing data may be worthwhile if we are about to base consequential decisions on that data. But we must be careful to propagate the changes back into the world, not merely touch up its reflection.

Technical Bias

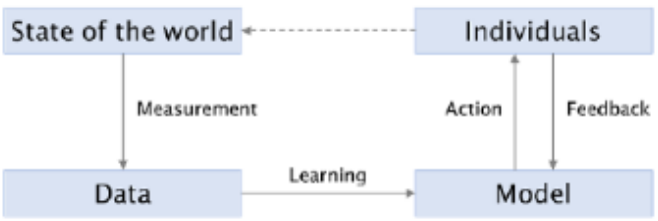
Arises from the technical operation of the system, potentially amplifying preexisting biases. Technical bias can happen at different phases of process.
During the **data cleansing**, methods for missing-value imputation based on incorrect assumptions may distort protected group proportions. For example, consider a form that gives applicants a binary gender choice but also allows gender to be unspecified. Suppose that half of the applicants identify as men and half as women, but that women are more likely to omit gender. Generally, missing-value imputation typically uses the most frequent classes as target variables, leading to a distortion for small groups, because membership in these groups will not be imputed. If a mode imputation is applied, then all (predominantly female) unspecified gender values will be set to male. This example illustrates that bias can arise from an incomplete or incorrect choice of data representation. It has been documented that data-quality issues often disproportionately affect members of historically disadvantaged groups, so we risk compounding technical bias due to data representation with bias due to statistical concerns.
During the **filtering**, selections and joins can arbitrarily change the proportion of protected groups even if they do not directly use the sensitive attribute as part of the predicate or the join key.

Emergent Bias

Develops during the system's use, influenced by feedback loops and user interactions. Recommendation systems in e-commerce creating a "rich-get-richer" effect, where popular items become more visible and thus more popular.

10.2 Bias in the ML loop

Where can technical and emergent biases occur in the machine learning (ML) loop?



Data Collection Phase

The initial phase involves transforming the state of the world into data through defined measurements. This process translates real-world complexities into structured datasets of rows, columns, and values. However, the term 'measurement' is somewhat misleading, as it suggests precision and objectivity—attributes that may not fully align with the inherently subjective decisions involved. These decisions can inadvertently perpetuate discrimination, particularly in relation to socially salient categories that have historically been the basis for unjust treatment.

Feature Representation

Features within a dataset may implicitly or explicitly encode an individual's status in a protected category. For example, a 'sensitive attribute' might be a discrete variable that represents one or several sensitive characteristics. Simply removing or ignoring these attributes does not guarantee the fairness of the classifiers built from the data. Features often exhibit minor correlations with sensitive attributes. For instance, visiting a specific website like Pinterest could be weakly correlated with female users, yet such correlations are generally too weak to accurately classify gender on their own. Moreover, removing a sensitive attribute could potentially disadvantage the individual it represents.

Measurement Challenges

Measurement is a nuanced process fraught with subjective decisions and technical challenges. For example, despite policies like Affirmative Action, an analysis by The New York Times in 2017 highlighted how Black and Hispanic students remain underrepresented at top colleges, more so than 35 years ago. Such comparisons often overlook the implications of newly introduced categories, like the 'multiracial' category in 2008.

Defining Target Variables

In ML, defining the target variable is a critical step that involves creating new categories. This stage is susceptible to biases which, if not addressed, will skew the model's predictions. The target variable is challenging to define accurately as it often represents a new construct rather than a well-understood, pre-existing one.

Training Data and Model Learning

The training data mirrors the disparities, distortions, and biases from both the real world and the measurement process. When training an ML model, it is crucial to discern whether the model preserves, mitigates, or exacerbates these disparities. For instance, certain real-world associations like smoking linked to cancer are valuable, while stereotypes such as gender-based color preferences are biases that should be avoided. Additionally, uniform sampling from diverse data can lead to underrepresentation of minority groups, affecting the model's performance on these groups.

Correlation versus Causation

A significant limitation of ML is its focus on correlations, which are often mistakenly interpreted as causations. This misunderstanding can lead to problematic applications of ML predictions.

Feedback Loops

The final phase of the ML loop involves interpreting feedback, which is complex due to:

- **Self-fulfilling Predictions**: For example, a predictive policing system might identify certain city areas as high-risk and increase police deployment there. This action could validate the initial prediction, regardless of whether it was based on biased data.
- **Impact on Training Data**: Further, arrests made in these areas could be added to the training dataset, perpetuating the notion that these areas are high-risk.

Assessment

In the assessment of the outcome we can't trust only on accuracy because misses some important aspects. For example, a classifier that always predicts no traffic fatality in the next year might have high accuracy on any given individual, simply because fatal accidents are unlikely. To make a correct evaluation we need to consider other criteria as recall and precision.

10.2.1 Mitigating Bias

To avoid bias we have to define some non-discrimination criteria. They defines the absence of discrimination in terms of statistical expressions involving random variables describing a classification scenario. They are properties of the joint distribution of the sensitive attribute A, the target variable Y, the classifier $\hat{Y}$ or score R and in some cases also features X. Researchers have proposed dozens of different criteria but main are the following:

- **Independence**: Random variables (A, R) satisfy independence if A $\perp$ R. If R is a score function that satisfies independence, then any classifier $\hat{Y} = \mathbf{1}\{R > t\}$ satisfies independence so long as the threshold is independent of group membership. Independence is often referred to as demographic parity, statistical parity, group fairness, disparate impact and others because define that probabilities among different categories must be the same.

$$\mathbb{P}\{\hat{Y} = 1 \mid A = a\} = \mathbb{P}\{\hat{Y} = 1 \mid A = b\}. \tag{10.1}$$

  To increase the flexibility of this criteria we can introduce a positive amount of slack e > 0.

$$\frac{\mathbb{P}\{\hat{Y} = 1 \mid A = a\}}{\mathbb{P}\{\hat{Y} = 1 \mid A = b\}} \geq 1 - \varepsilon. \tag{10.2}$$

  Independence is often the easiest one to work with mathematically and algorithmically, but decisions based on it can have undesirable properties: imagine a company that in group a hires diligently selected applicants at some rate p > 0 and in group b, the company hires carelessly selected applicants at the same rate p. Even though the acceptance rates in both groups are identical, it is far more likely that unqualified applicants are selected in one group than in the other. It will appear in hindsight that members of group b performed worse than members of group a, thus establishing a negative track record for group b. That can take to the **Glass cliff** effect, where women and people of color are more likely to be appointed CEO when a firm is struggling.
- **Separation**: Random variables (R,A,Y) satisfy separation if R $\perp$ A | Y. That means that have the same accuracy and the same rate of errors on the different categories.

$$\mathbb{P}\{\hat{Y} = 1 \mid Y = 1, A = a\} = \mathbb{P}\{\hat{Y} = 1 \mid Y = 1, A = b\} \tag{10.3}$$
$$\mathbb{P}\{\hat{Y} = 1 \mid Y = 0, A = a\} = \mathbb{P}\{\hat{Y} = 1 \mid Y = 0, A = b\} \tag{10.4}$$

  The idea of equalizing error rates across has been subject to critique because an optimal predictor need not have equal error rates in all groups. Enforcing equality of error rates leads to a predictor that performs worse in some groups than it could be.
- **Sufficiency**: The random variables (R, A,Y) satisfy sufficiency if Y $\perp$ A | R. We recognize this condition as requiring a parity of positive/negative predictive values across all groups.

$$\mathbb{P}\{Y = 1 \mid R = r, A = a\} = \mathbb{P}\{Y = 1 \mid R = r, A = b\}. \tag{10.5}$$

Metric	Definition	Fairness notion	Range	Interpretation
Disparate Impact (*DI*) [29]	$\frac{Pr(\hat{Y}=1 \mid S=0)}{Pr(\hat{Y}=1 \mid S=1)}$	demographic parity	$[0, \infty)$	$DI = 1 \rightarrow$ completely fair $DI = 0 \rightarrow$ completely unfair $DI = \infty \rightarrow$ completely unfair
True Positive Rate Balance (*TPRB*) [41]	$Pr(\hat{Y}=1 \mid Y=1, S=1) - Pr(\hat{Y}=1 \mid Y=1, S=0)$	equalized odds	$[-1, 1]$	$\lvert TPRB \rvert = 0 \rightarrow$ completely fair $\lvert TPRB \rvert = 1 \rightarrow$ completely unfair
True Negative Rate Balance (*TNRB*) [41]	$Pr(\hat{Y}=0 \mid Y=0, S=1) - Pr(\hat{Y}=0 \mid Y=0, S=0)$	equalized odds	$[-1, 1]$	$\lvert TNRB \rvert = 0 \rightarrow$ completely fair $\lvert TNRB \rvert = 1 \rightarrow$ completely unfair

How to satisfy a non-discrimination criterion? We distinguish between three different techniques.

- **Pre-processing approaches**: These approaches modify the data before training to remove biases, which subsequently ensures that the predictions made by a learned classifier satisfy the target fairness notion. Are motivated from the fact that ML techniques are data-driven and the predictions of a classifier reflect trends and biases in the training data. The main advantage of pre-processing is that it is model-agnostic, allowing flexibility in choosing the classifiers based on the application requirements.
- **In-processing approaches**: It takes place within the training stage and fairness is typically added as a constraint to the classifier's objective function. The advantage of in-processing lies precisely in the ability to adjust the classification objective to address fairness requirements directly, and, thus has the potential to provide guarantees. However, in-processing techniques are model-specific and require re-implementation of the learning algorithms to include the fairness constraints.
- **Post-processing approaches**: They are model-agnostic and enforce fairness by manipulating predictions made by an already-trained classifier. Their benefit is that they do not require classifier retraining. However, since post-processing is applied in a late stage of the learning process, it offers less flexibility.

11. MLOPS

Machine learning involves creating algorithms that can learn and improve from experience without explicit programming for specific tasks. These algorithms analyze training data to build models that can make predictions on new, unseen data. ML is used in various applications, from spam detection to image recognition, because it can handle complex and evolving problems better than traditional programming.

11.1 When is Machine Learning useful?

On one hand, machine learning can be the best solution in the following cases:

1. **Complex Problems**: ML is ideal for tasks that are too complex for traditional coding. For example, spam detection systems use ML to identify patterns and characteristics of spam emails, which would be infeasible to hard-code.
2. **Constantly Changing Problems**: ML is useful for problems that change over time, requiring regular updates to the programming logic. For instance, recommendation systems need to adapt to new user preferences and trends continuously.
3. **Perceptive Problems**: ML excels at detecting patterns in data that might not be immediately obvious. This is useful in fields like medical diagnosis, where subtle patterns in medical images can indicate diseases.
4. **Unstudied Phenomena**: ML can predict outcomes in areas where the phenomena are not well understood but data is available. An example is analyzing logs from a complex system to predict failures.
5. **Cost-Effectiveness**: The cost of implementing ML includes data collection, preparation, model training, and infrastructure for serving and monitoring the models. ML is cost-effective when these components are cheaper than the manual or traditional programming alternatives.

On the other hand, it might not be suitable when:

- **Explainability Requirements**: In situations where every decision must be explainable, such as in legal or medical fields, ML's black-box nature can be a drawback.
- **High Error Cost**: If the cost of errors is prohibitively high, ML's probabilistic predictions might not be acceptable.

- **Speed to Market**: Developing ML models can be time-consuming. When rapid deployment is crucial, simpler solutions might be preferable.
- **Data Challenges**: If obtaining the right data is difficult or if simple heuristics can achieve acceptable performance, ML might be unnecessary.
- **Exhaustive Systems**: For systems that do not require frequent updates or improvements, traditional methods like lookup tables might suffice.

11.2 ML Project Failures

According to various estimates made the 85% of big data projects fail (Gartner, 2017) and the 87% of data science projects never make it to production. What are the reasons? This can be due to overestimation of ML capabilities or underestimation of the challenges involved.

1. **Lack of Experienced Talent**: There are few standardized ways to teach data science and ML engineering, leading to a shortage of experienced professionals.
2. **Leadership Support**: Successful ML projects require support from leadership, which might not always align with the high-uncertainty nature of scientific work.
3. **Data Infrastructure**: The success of ML projects heavily relies on the quality and availability of data. Poor data infrastructure can hinder project progress.
4. **Data Labeling**: A significant portion of project time is spent on custom data labeling, which can be prone to quality issues. Most of time labeling is done by hand by data scientist that are not experts of domain of the project.

11.3 Machine Learning Operations

MLOps, or Machine Learning Operations, encapsulates the methodologies used to seamlessly transition ML models from research to robust, real-world applications. It ensures that models are deployed and maintained in production settings in a reliable and efficient manner, effectively converting experimental ML models into tools that deliver tangible business value.
Consider the scenario where an ML model is developed in a controlled setting such as Jupyter notebooks; this environment vastly differs from a production scenario. Deploying a model into production involves its integration with the existing codebase and the ability for other systems to utilize the model's predictions.
The developmental lifecycle of an ML model includes variations in data, models, and associated code. For instance, a model might initially perform well post-deployment, but over time it may degrade or exhibit unexpected behaviors. This degradation could necessitate gathering new data to retrain the model. During this process, it might become apparent that acquiring the specific data needed to address the original problem is challenging, prompting a reevaluation or reformulation of the problem. Additionally, post-deployment feedback may reveal inaccuracies in the initial assumptions used during the model's training phase, necessitating model adjustments. Furthermore, shifts in business objectives during the project's lifespan might lead to changes in the chosen machine learning algorithms, underscoring the need for adaptability in model development. Several issues can arise in the MLOps process:

- **Data Quality**: ML models are highly sensitive to the quality, semantics, and completeness of data.
- **Model Decay**: Over time, the performance of ML models can degrade as the underlying data distributions change.
- **Locality Issues**: Pre-trained models might not perform well across different user demographics due to variations in data.
- **Need for MLOps**: Establishing robust processes around designing, building, and deploying ML models into production is essential for sustained performance and value.

Pushing machine learning models into production without MLOps infrastructure is risky: the performance of a machine learning model can often be only assessed in the production environment; in addition, pushing models into production is not the final step of the machine learning life cycle far from it.
MLOps is composed of five key components:

1. **Development**: This stage involves data engineering, model training, evaluation, and packaging. Data engineering includes collecting, cleaning, and preparing data for training. Model training involves selecting algorithms, feature engineering, and hyper-parameter tuning. Evaluation ensures that the models meet performance objectives.
2. **Deployment**: Deploying models into business applications involves serving the models, monitoring their performance, and logging results for further analysis.
3. **Monitoring**: Continuous observation of model performance to manage resource usage and detect issues early.
4. **Iteration**: Regularly updating models based on new data and performance metrics to ensure they remain accurate and relevant.
5. **Governance**: Ensuring compliance with regulations, maintaining security standards, and upholding ethical considerations in ML model deployment and use.

11.3.1 Development

The phase of development involves several practices:

- **Data Engineering Pipeline**: This includes the processes of collecting, preparing, and splitting data for training ML models. Proper data engineering is crucial for building robust models. The data engineering pipeline includes the following techniques:
  - **Data Ingestion**: Collecting data by using various frameworks and formats. This step might also include synthetic data generation or data enrichment. More in detail data ingestion includes data sources identification, storage estimation, obtaining data, backup data, privacy compliance, metadata catalog (documenting the metadata by recording the basic information about the size, format, aliases), test data. During this step is important to understand what each attribute exactly represents to avoid data leakage, that is the unintentional introduction of information about the target that should not be made available (target being a function of a feature, feature hiding the target, feature coming from the future). Dealing with data is not easy for the following reasons: getting unlabeled data can be expensive, we have to manage bad quality data and noise and avoid bias. Those can be of different types:
    1. Selection bias is the tendency to skew your choice of data sources to those that are easily available, convenient, and/or cost-effective.
    2. Self-selection bias is a form of selection bias where you get the data from sources that "volunteered" to provide it. Most poll data has this type of bias.
    3. Omitted variable bias happens when your featurized data doesn't have a feature necessary for accurate prediction.
    4. Sampling bias (also known as **distribution shift**) occurs when the distribution of examples used for training doesn't reflect the distribution of the inputs the model will receive in production.
    5. Prejudice or stereotype bias
    6. Experimenter bias is the tendency to search for, interpret, favor, or recall information in a way that affirms one's prior beliefs or hypotheses.
    7. Labeling bias happens when labels are assigned to unlabeled examples by a biased processor person
  - **Exploration and Validation**: Includes data profiling to obtain information about the

content and structure of the data. The output is a set of metadata, such as max, min, avg of values. Data validation operations are user-defined error detection functions, which scan the dataset to spot some errors.
  - **Data Wrangling (Cleaning)**: The process of re-formatting particular attributes and correcting errors in data, such as missing values imputation, fixing outliers and removing not relevant data. During this step we have to deal with **class imbalance**, that is a very uneven distribution of labels in the training data. To manage it we have two main techniques:
    1. **Oversampling** by making multiple copies of minority class examples, increasing their weight (SMOTE).
    2. **Undersampling** by removing from the training set some examples of the majority class (Tomek links).
  - **Data Labeling**: The operation of the Data Engineering pipeline, where each data point is assigned to a specific category.
  - **Data Splitting**: Splitting the data into training, validation, and test datasets to be used during the core machine learning stages to produce the ML model
- **Training and Evaluation**: Training involves selecting and optimizing algorithms. Evaluation ensures that the model's performance is satisfactory and meets predefined criteria. In this phase we, also, have to define how package the model that means select a specific format to be consumed by the business application.
  1. **Error analysis**: categorize/tag the mistakes of the model to understand the reasons.
  2. **Cross-checking** between different metrics, that means checking how the model reacts to different inputs to understand if the model can reach at least the human level performance.

Type	Accuracy	Human level performance	Gap to HLP	% of data
Clean Speech	94%	95%	1%	60% → 0.6%
Car Noise	89%	93%	4%	4% → 0.16%
People Noise	87%	89%	2%	30% → 0.6%
Low Bandwidth	70%	70%	0%	6% → ~0%

Figure 11.1: Cross-checking technique

- **Reproducibility**: Keeping track of model versions and configurations to ensure that results can be reproduced. This involves version control of code, data, and model parameters.
- **Responsible AI**: Explainability techniques offer a way to mitigate uncertainty and help prevent unintended consequences.

11.3.2 Preparing for production

Before deploy a model to production we have to make sure it's technically possible.

- **Adaptation**: Ensuring that models are compatible with production environments. Two main extreme scenarios are possible:
  1. The development and production platforms are from the same vendor or interoperable (only few clicks needed).

2. The model needs to be reimplemented from scratch possibly by another team, and possibly in another programming language

- **Performance Considerations**: Understand computational performance and resource usage.
- **Model Risk Evaluation**: Anticipating potential risks associated with model deployment and taking steps to mitigate them. ML model risk originates essentially from bugs, low quality of training data, expected error rates, misuse of the model or misinterpretation of its outputs, adversarial attack, legal risk and reputational risk due to bias, unethical use of machine learning.

11.3.3 Deployment

We need to deploy a model as part of a business application such as a mobile or desktop application. So the following steps are required:

- Model Serving: the process of addressing the ML model artifact in a production environment.
- Model Performance Monitoring: The process of observing the ML model performance based on live and previously unseen data.
- Model Performance Logging: Every inference request results in the log-record

During this phase all the techniques defined by the DevOPS can be applied to ML project:

- **CI/CD Pipelines**: Automating the processes of building, testing, and deploying ML models. This ensures a smooth transition from development to production.
- **Artifacts**: Bundles of code, data, and documentation required for deploying models. These artifacts ensure that the deployment process is consistent and reliable.
- **Testing**: Validating model properties and ensuring robustness against various data scenarios. The tests should make it as easy as possible to diagnose the source issue of failure. In particular we have to develop two types of test: A "base case" test on a fixed dataset with simple data and not-too- restrictive performance thresholds and a number of datasets each with a specific oddity (missing values, extreme values, etc.)

To deploy the model in the right way we have to adopt a specific strategies. We have many options:

- **Blue-green/red-black deployment**: A new system is set up next to the stable one; when it's functional, the workload is directed to the newly deployed version. If it remains healthy, the old one is shut down.
- **Canary releases**: The stable version of the model is kept in production. A percentage of the workload is redirected to the new model, and results are monitored. In this way, a malfunction would likely impact only a small portion of the workload (any malfunction is an incident, but the idea is to limit the blast radius).
- **Shadow model deployment**: The ML system runs in parallel and shadows the human. The ML outputs are at the beginning not used for any decision. Only when we are confident of the results.

Managing the versions of a model requires to provide an exact description of the environment. Containerization technology is increasingly used to tackle these challenges (Docker).

11.3.4 Monitoring

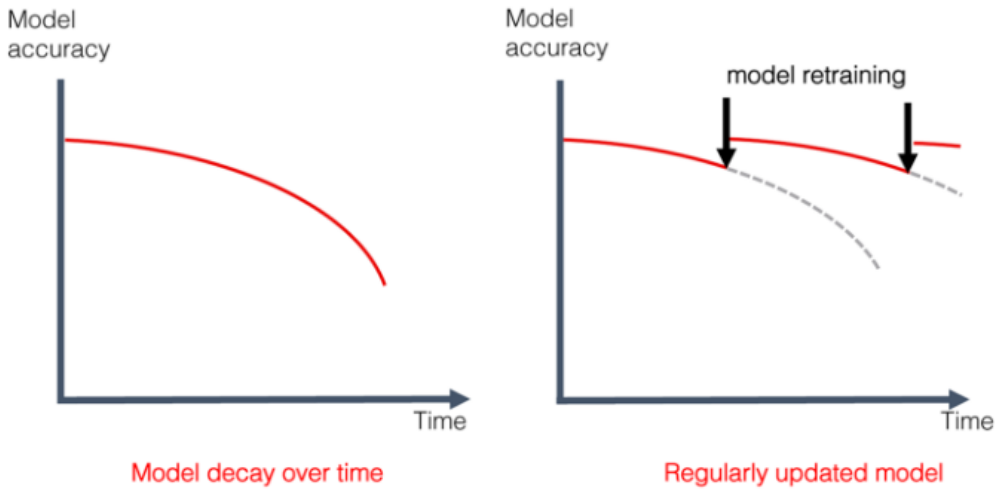
At a high level, there are three maintenance measures:

- **Resource Monitoring**: Tracking usage of computational resources like CPU, memory, disk, and network.
- **Health Checks**: Regularly querying the model to ensure it behaves as expected and is available for predictions.
- **ML Metrics Monitoring**: Analyzing model accuracy and detecting signs of staleness or performance degradation. This can trigger retraining if necessary.

If a malfunction is detected, a rollback to a previous version may be necessary.

Handling Model Decay

A common problems of model in production is the **Model decay**: past performance of ML systems is no guarantee of future results. We have two main drift that can occur in the model:



- **Data Drift**: Changes in the input data distribution that impact model relevance. This requires monitoring and potentially retraining the model.
- **Concept Drift**: Changes in the relationships between model inputs and outputs. This can occur due to changes in the underlying processes that generate the data.
  - **Gradual Concept Drift**: Gradual or incremental drift is the one we expect. The world changes and the model grows old (competitors launch new products or macroeconomic conditions evolve).
  - **Sudden concept drift**: External changes might be more sudden or drastic (e.g. COVID19)
  - **Recurring concept drift**: describe repeating changes as seasonality.

Machine learning models require diligent monitoring across two primary dimensions: resource utilization (including CPU, RAM, and network usage) to ensure seamless operation within the production environment, and model performance to gauge the continuing relevance and accuracy of the model over time.

To effectively track changes, we can rely on a variety of metrics:

- **Software Metrics**: These include memory usage, compute resources, latency, throughput, and server load, which help assess the operational efficiency of the model.
- **Input Metrics**: These metrics, such as average input length, input volume, or the number of missing values, offer insights into the quality and consistency of the data being fed into the model.
- **Output Metrics**: Metrics like the frequency of null returns from the system provide clues about the model's output reliability and decision-making capabilities.

Post-decay, models often require retraining to maintain accuracy and efficacy. The frequency of this retraining is influenced by factors like the specific domain, associated costs, and initial model performance. Regular retraining cycles enable the model to adapt to new data and conditions. The schedule for these cycles should consider both the availability of ground truth data, which sets the minimum frequency, and the capabilities of the team managing the models, which establishes the maximum feasible frequency.

Monitoring ground truth data is essential but can be challenging. It may not always be immediately accessible, and there is often a need to synchronize this data with the observed predictions. Addition-

ally, obtaining comprehensive ground truth data can be cost-prohibitive, as it may require extensive manual verification processes. A typical scenario is fraud detection, where each transaction might need individual examination, making the process time-consuming and complex.

Bibliography