DATA MINING

by Yannick Giovanakis

August 31, 2018

Contents

1 Introduction

Digital data storing is a task that started in the early '60s and became since then an important field of computer science. With the passing of time technologies became more powerful and storing large amounts of data became easier.
Analysing bigger and bigger amounts of data became a difficult task over time so **automation** was required . In the late '80s / start of '90s **data mining** became an important task to make sense and interpret massive amounts of data and has been a crucial since then in many fields (customer attrition,credit assessment ,customer segmentation, community detection and many more) as part of a larger subject called **data science**
Data mining is the non-trivial process of identifying

- **valid**
- **novel/interesting**
- **useful**
- **understandable**

patterns in data that results in some **worthy** information.
The aim of data mining is to build programs that run automatically on large databases to seek for regularities or patterns. The main problem is that most patterns are **uninteresting, spurious,inexact** and based on real **imperfect** data. This is why data mining algorithms need to be **robust** to cope with imperfections and to extract regularities that are inexact but **useful**.
The informations found can then be used for:

- **predictive** tasks → create predictions based on patterns
- **descriptive** tasks → create insights based on patterns
- **prescriptive** tasks → combination of both

1.1 Data mining process

1. Interesting question
   What is the goal? What must be predicted?
2. Get the data
   How was data sampled ? Which data is relevant?
3. Explore the data
   Plot data,compute statistics , search for anomalies
4. Build model
   Build,fit ,validate
5. Communicate result
   What did we learn? Is there a result?

The data part is the most important :

- **Selection**
  What are the data we actually need?
- **Cleaning**
  Are there errors / inconsistencies that need to be eliminated?
- **Transformation**
  Some data can be eliminated because equivalent to other data or used to get new data
- **Mining**
  Select mining approach : **classification ,regression ,clustering...** and apply algorithms
- **Validation**
  Are the patterns found **sound** ? According to which criteria? Can the results be explained?

1.2 Data mining tasks

- Prediction & Regression
- Classification
- Clustering
- Association rules
- Trend & evolution analysis
- Outlier analysis
- Text mining ,topic modelling, graph mining

1.3 Issues

Data mining generates many patterns, but typically only **few** are interesting. It is important to find an **interestingness** measure : a pattern is interesting if it is understood, valid on new data/test data with some degree of certainty, potentially useful,novel or validates some hypothesis that needed confirmation.
Interestingness measures can be **objective** (based on **statistics** and **patterns**) or **subjective** (based on **belief** in data) .
Can **all** interesting patterns be found?

Completeness problem

Can a data mining system find **all** the interesting data within a dataset? This depends also on the data mining approach that has been chosen.

Optimization problem

The data mining system should **only** find useful and interesting patterns , by either filtering **all possible patterns** or by using **mining query optimization**

2 Data representation

Inside databases and datasets we can identify:

- **Instances** → observations/cases/records that represent atomic elements of information
- **Attributes** → variables/features that measure aspects of an instance. Each instance has a certain number of attributes
- **Concepts** → things that can be learned inside the data

2.1 Attribute types

Data types are not only important for our understanding but some algorithms work better with a certain type of data : make it easier to make adequate comparisons for example.
Knowing the data type is also important the check for **valid values** and deal with **missing values**.

Numeric attributes

- Real-valued or integer-valued domain
- Interval-scaled when only differences are useful → temperature
- Ratio-scaled when only rations are meaningful → age

Numerical attributes are **ordered** and measured in fixed units.
Zero point is only defied for **ratio attributes**.
Can be divided into *discrete or continuous*.

Categorical attributes

- Set-valued domain composed of a set of symbols
- *Nominal*
  When only equality is meaningful Values are distinct symbols that serve

as labels. No relation is implied among nominal values and only equality test can be performed.

- *Ordinal*
  When both equality and inequality are meaningful. As in the nominal case, also here talking about **difference** doesn't make sense.

Binary attributes
Represented by either 0/1

Sometimes the same attribute can be either considered **nominal** or **ordinal** : `if age == young AND ...` is nominal whereas `if age < presbyopic AND...` is ordinal.

2.2 Missing values

Many databases present **missing values** . The nature of missing values is very broad and must be understood : it can be due to faulty equipment, incorrect measurements, censored or anonymous data ecc...
Missing values can have their own importance (ex: a missing test) but in that case is should have its own coding.It is important to find out **why** the value is missing : if the absence of value has some significance then **'missing'** is a separate value; if it does not it must be treated in a **special way**. Usually missing values are indicated by **out of range** entries, **Nan** or **special values**.

2.2.1 Missing values - types

There are three types of missing values:

- **MCAR - Missing completely at random**
  The distribution of an example having a missing value for an attribute **does not depend on either the observed data or the missing data** (doesn't give you a hint on why it is missing). Example : people omitting geolocation on Twitter.

- **MAR - Missing at random**
  The distribution of an example having a missing value for an attribute **depends on the observed data, but does not depend on the missing data** (the missing is not related to the missing data but to some observed data) . Example : people not giving out how much they earn not because of the amount but because they don't want to.

- **NMAR - Not missing at random**
  The distribution of an example having a missing value **depends on the missing value**. Example : people not telling they salary because of their position (so related to the amount).

2.2.2 Missing values - solutions

When dealing with missing values one should always try to find out

- Why data is missing
- Distribution of missing data

Then different methods can be applied to deal with missing values depending on the type of missing value:

- **Discard** examples with missing values
  Simplest approach which allows to used **unmodified data**. This method should be used only if we have **lots of data** and with **low number** of missing values because otherwise **bias** can be introduced.

- **Fill in manually** missing values

- **Convert into new value**
  Using a special value just for the missing value one then you can use a boolean attribute to tell if it is missing or not (increases difficulty of data mining process!)

- **Imputation**
  Assign a value to the missing one based on the rest of the dataset (mean/-mode substitution, dummy variable imputation, regression...)

Some methods :

- **Listwise deletion**
  This way you only analyse cases with available data on each variable. This can reduces the number of available data greatly and should only be used with MCAR types in order to not **introduce bias**

Gender	8th grade math test score	12th grade math score
F	45	.
M	.	99
F	55	86
F	85	88
F	80	75
.	81	82
F	75	80
M	95	.
M	86	90
F	70	75
F	85	.

- **Pairwise deletion**
  This way examples are deleted only if considering that specific attribute (say you are considering the Gender attribute then you only discard row 5). The method keeps as many cases as possible for each analysis but does not allow for comparison as the sample size can vary.

Gender	8th grade math test score	12th grade math score
F	45	.
M	.	99
F	55	86
F	85	88
F	80	75
.	81	82
F	75	80
M	95	.
M	86	90
F	70	75
F	85	.

- **Imputation**
  Imputation extracts a model from available data. It is suitable for MCAR and MAR (to a lesser extend) but not for NMAR (for NMAR we need to go back to the source of the data to obtain more information). There a several imputations methods like:

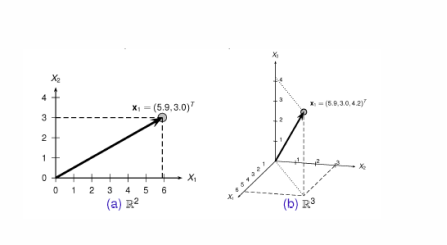
  - **Mean/mode substitution**
    Replace the missing values with the mean/mode . This allows to use the **complete case analysis** methods but **reduces variability**
  - **Dummy variable control**
    Create an indicator for missing value (1 = missing, 0 =not missing) and impute missing values to constant (such as the mean). Then you include the missing indicator in the algorithm. Unfortunately results in **biased estimates**
  - **Regression**
    Replaces values with predicted values from a regression equation.
    Otherwise you can choose to **not impute** by simply using the default policy of the data mining methods (if one is present!)

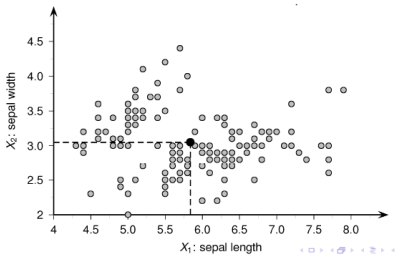
2.3 Inaccurate values

Inaccurate values can happen for many reasons. For example data being collected but not for mining purposes : in this case an error does not affect the original purpose of the data (for example age of customer). Errors can be of many types :

- **Typographical errors in nominal attributes**
- **Typographical errors in numeric attributes**
- **Deliberate errors** (invent answers for example)

Other than the **probabilistic view** of data (distribution,mean...) the more common way to view data is the **geometric view**.





2.4 Data format

There are not many data formats (neither for storing nor for data exchange) as most commercial tools have their own format.Most tools import **excel** and **CSV - comma separated values** files. Over the years many **standardization formats** have been proposed :

- **Attribute Relation File Format (ARFF)**
  Very popular ,rising since Java ML libraries have been made available. With the `@attribute` you can declare an attribute name and specify its **type** ,**possible values** or **range**. This allows for **cross checking** if the data is correct.
  ARFF also supports **sparse representation**

$$0, 26, 0, 0, 0, 0, 63, 0, 0 \rightarrow \{1 \quad 26, 6 \quad 63, 10\}$$

  Missing values in ARFF are **'?'**

```
@attribute 'bereavement-assistance' {'yes','no'}
@attribute 'contribution-to-health-plan' {'none','half','full'}
@attribute 'class' {'bad','good'}
@data
1,5,?,?,40,?,?,2,?,1 1,'average',?,?,'yes',?,'good'
2,4.5,5.8,?,?,35,'ret_allw',?,?,'yes',1 1,'below_average',?,'full',?,'full','good'
?,?,?,?,?,38,'empl_contr',?,5,?,1 1,'generous','yes','half','yes','half','good'
3,3.7,4,5,'tc',?,?,?,?,'yes',?,?,?,?,'yes',?,'good'
```

- **Dataset Publishing Language (DSPL)**
  Open format from Google. Simply add an XML file to an existing CSV file

2.5 Model representation

The **Predictive Model Markup Language** is an XML-based markup language created to provide a way for applications to define models related to predictive data mining.The goal is to **share** models between applications using different components (data dictionaries, data transformations,mode,data mining schema,target,output)

3 Data Exploration

It's the preliminary exploration of the data aimed at **identifying their most relevant characteristics**. In this step many important information inside the data can be found like outliers, missing values , data ranges etc. which are crucial to pick the right preprocessing and data mining steps.

3.1 Summary statistics

Summary statistics are numbers that summarize properties of data : location,mean,spread,skewness ,standard deviation ,mode, percentiles, etc.

- **Frequency**
  Percentage of time value occurs in the data set (used with categorical data)
- **Mode**
  Most **frequent** attribute value (used with categorical data)
- **Mean**
  Most common measure of the location of a set of points. It is **very sensitive** to outliers.
  $$\text{mean(x)} = \bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$$
- **Median**
  The median is the value separating the higher half from the lower half of a data sample.
  $$\text{median(x)} = \begin{cases} x_{r+1} & \text{if } m \text{ is odd , like m=2r+1} \\ \frac{1}{2}(x_r + x_{(r+1)}) & \text{if } m \text{ is even , like m=2r} \end{cases}$$
- **Percentiles**
  Given an ordinal or continuous attribute x and a number p between 0 and 100 the p-th percentile is a value x_p of x such that $p\%$ of the observed values of x are less than x_p

- **Trimean**
  Is the weighted mean of the first , second and third quartile :

$$TM = \frac{x_{25} + 2x_{50} + x_{75}}{4}$$

- **Truncated mean**
  Discards data **above** and **under** a certain percentile.

- **Interquartile mean**

$$X_{IQM} = \frac{2}{n} \sum_{i=0.25n+1}^{0.75n} x_i$$

- **Range**
  Difference between max and min

- **Variance**

$$\text{variance(x)} = s_x^2 = \frac{1}{m-1} \sum_{i=1}^{m} (x_i - \bar{x})^2$$

  . Sensitive to outliers (see below for robuster statistics)

- **Average absolute deviation**

$$AAD(x) = \frac{1}{m} \sum_{i=1}^{m} |x_i - \bar{x}|$$

- **Mean absolute deviation**

$$MAD(x) = \text{median}\left(\{|x_1 - \bar{x}|, ..., |x_m - \bar{x}|\}\right)$$

- **Interquartile range**

$$\text{interquartile range(x)} = x_{75\%} - x_{25\%}$$

- **Correlation analysis**
  Given two attributes it measures how strongly one attribute implies the other ,based on available data (**does not imply causation**).

- **Numerical values**
  Compute the c**orrelation coefficient**, Pearson's product
- **Ordinal variables**
  Compute **Spearman** rank correlation coefficient
- **Categorical variables**
  Compute the χ^2 **statistic test** which tests the hypothesis that A and B are independent.
- **Binary variables**
  Compute **point-biserial** correlation

- **Outliers**
  They are data objects that do not comply with the general behaviour or model of data,that is , values that appear as **anomalous**. Most data mining methods consider outliers **noise** or **exceptions**. They can be detected using :
  - Statistical tests that assume a distribution or probability model for the data
  - Distance measure where objects that are substantial distance from any other cluster are considered outliers
  - Deviation based models identify outliers by examining differences in the main characteristics of the objects in a group

  They are usually **filtered out by trimming** or replaced with the **Winzorsing technique** : a 10% Winsorizing considers $5th$ and 95^{th} percentiles , setting values below the 5^{th} to the 5^{th} itself and above the 95^{th} to the 95^{th} itself.

3.2 Normalisation

When attributes have vastly different **scales** (e.g. age vs income) it is necessary to normalize them.

- **Range normalisation**

$$x_i' = \frac{x_i - min_i x_i}{max_i x_i - min_i x_i}$$

- **Standard score normalisation**

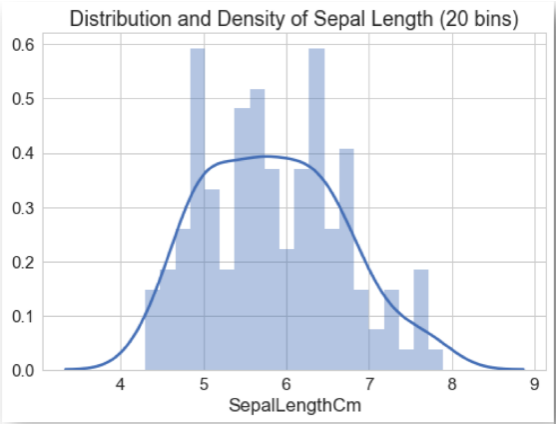
$$x'i = \frac{x_i - \mu}{\sigma}$$

3.3 Visualisation

Visualisation is the conversion of data into **visual** or **tabular** format so that the characteristics of data and relationships among data points can be analysed (detected **patterns,trends,outliers...**)

- **Bar plot**
  They use horizontal or vertical bars to compare categories. One axis shows the **compared categories** the other axis represents a **discrete value**. Variants are **grouped bar graphs** (clustered in groups of more than one) and **stacked bar graphs** (bars divided into subparts to show cumulative effect)

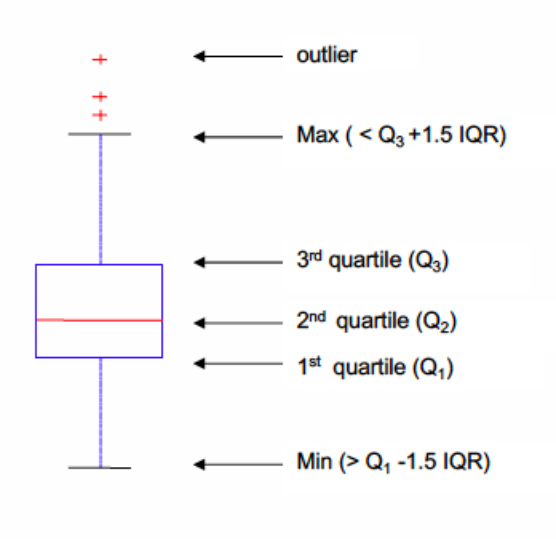
- **Histograms**
  They are a graphical representation of **distribution of data**. They estimate the probability distribution of continuous variables depicting adjacent rectangles whose areas are proportional to the frequency of the observations in the interval. The height of each bar indicates the number of objects



- **Scatter plots**
  Attributes values determine the position inside the scatter plot. Usually

scatter plots are two dimensional but also three dimensional scatter plots exist.

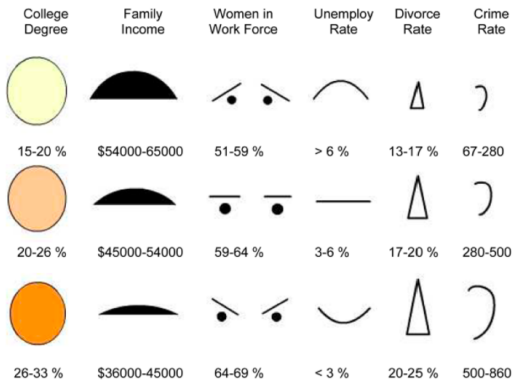
- **Box plots**
  They represent another way of representing the distribution of data



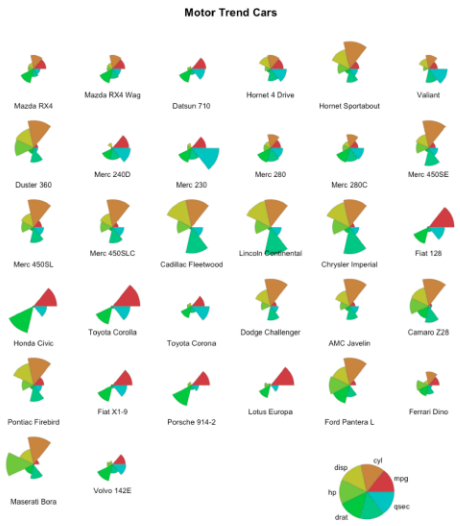
3.3.1 Multiple dimension at once visualisation

When dealing with more than 2 dimensions some plots allow to represent the data showing all the dimensions at once:

- **Chernoff faces**
  Associates each attribute with a characteristic of a face. The values of each attribute determine the appearance of the corresponding facial characteristic : each sample becomes a **separate face**.



- **Star plots**
  Similar approach that uses axes radiate from a central point.The line connecting the values of an object is a polygon



3.3.2 Dimensionality reduction

- **PCA**
  Applied to reduce the number of dimensions of data (**feature selection**). The goal of PCA is to find a projection that captures the **largest amount of variation** in data. Given N data vectors from n-dimensions find k¡n orthogonal vectors (principal components) that can be used to represent data. Works for numerical data if **affected by scale**

1. Normalize data
2. Compute k orthonormal unit vectors (**principal components**)
3. Each input vector is a **linear combination** of the k principal component vectors
4. The PC are sorted in order of **decreasing significance**

- **t-Distributed Stochastic Neighbour Embedding**
  It is a **non-linear** dimensionality reduction technique used to map high dimensionality data to **two or three dimensions**. The algorithm converts **similarities** between data points to **joint probabilities** and model high dimensional points into map points so that position of the map points aims at conserving the structure of the data (similar points are near, dissimilar points are far away)

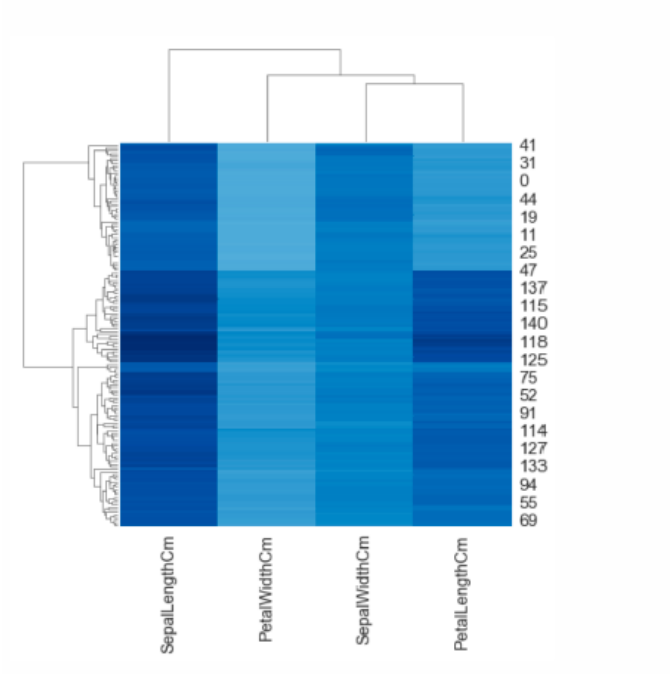
  1. Define a probability distribution over pairs of high dimensional data points so that **similar points** have high probability of being picked and **dissimilar points** have extremely small probability of being picked
  2. Define a similar distribution over the points of the map space by minimizing (**gradient descent**) the distance of the **Kullback-Leibler** divergence between that two distributions with respect to the locations of the map points.

  Different initialisation will lead to different results and should be applied to data with a reasonable number of dimensions (30-50). With more dimensions other algorithms should be applied.

3.3.3 Other visualisation techniques

- **Contour plots**
  Useful for continuous attributes measured on a spatial grid. They partition the plane into regions of similar values. The contour lines that form the boundaries of these regions connect points with equal values.

- **Heat maps**


41
31
0
44
19
11
25
47
137
115
140
118
125
75
52
91
114
127
133
94
55
69
SepalLengthCm
PetalWidthCm
SepalWidthCm
PetalLengthCm

4 Association Rules Mining

Association rules are **frequent** patterns,associations,correlations or causal structures among sets of items or objects in databases. They are often used in basket data analysis,cross-marketing, catalog design and or any other application where matching similar or dissimilar items together based on an underlying relationship is important.
Given a set of transactions the aim is to **find rules** that will **predict** the occurrence of an item based on the occurrence of **other** items in the transaction. Inside a **set of transaction** one can identify :

- **k-Itemsets**
  An k-itemset is a collection of k items (e.g.$\{bread, milk, jam\}, k = 3$)
- **Support**
  Fraction of transactions that contain an itemset (e.g. in a set of 8 transactions 3 contain itemset $\{bread, milk, jam\}$ so support is $\frac{3}{8}$)
- **Support count**
  Frequency of occurrence of an itemset $\sigma(\{Milk, Bread\}) = 3$
- **Frequent itemset**
  An itemset whose support is **greater than or equal to minsup threshold**.

This way an **association rule** can be defined as an implication of the form $X \implies Y$ where X,Y are **itemsets**. These rules can be evaluated by using

- **Support s**
  Fraction of transactions that contain both X (here Bread) and Y (here Milk)
  $$s = \frac{\sigma(\{Bread, Milk\})}{\text{num of transactions}}$$
- **Confidence c**
  Measures of often items Y appear in transactions that have X
  $$c = \frac{\sigma(\{Bread, Milk\})}{\sigma(\{Bread\})}$$

The goal is to find , given a set of transactions T , associations rules having:

- Support $\geq$ minsup threshold
- Confidence $\geq$ minconf threshold

To find associations rules **brute force** approach should **not** be used as it becomes easily computationally prohibitive.

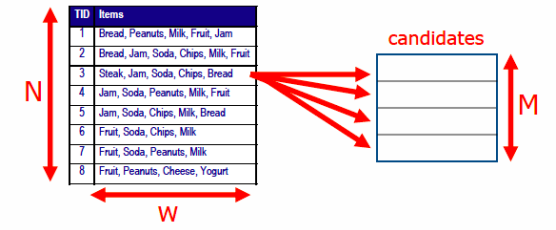
{Bread, Jam} $\Rightarrow$ {Milk} s=0.4 c=0.75
{Milk, Jam} $\Rightarrow$ {Bread} s=0.4 c=0.75
{Bread} $\Rightarrow$ {Milk, Jam} s=0.4 c=0.75
{Jam} $\Rightarrow$ {Bread, Milk} s=0.4 c=0.6
{Milk} $\Rightarrow$ {Bread, Jam} s=0.4 c=0.5

As seen in the figure above the rules all have same support as they are different partitions but with different confidence levels. This allows to **decouple** support and confidence:

1. Generate all itemsets whose support is $\geq$ minsup threshold (**Frequent itemset generation**).
2. Generate high confidence rules from frequent itemset where each rule is a binary partition of a frequent itemset (**Rule generation**).

4.1 Frequent itemset generation

Given d items there are 2^d possible **candidate datasets** which results in a huge complexity $\sim O(NMw)$



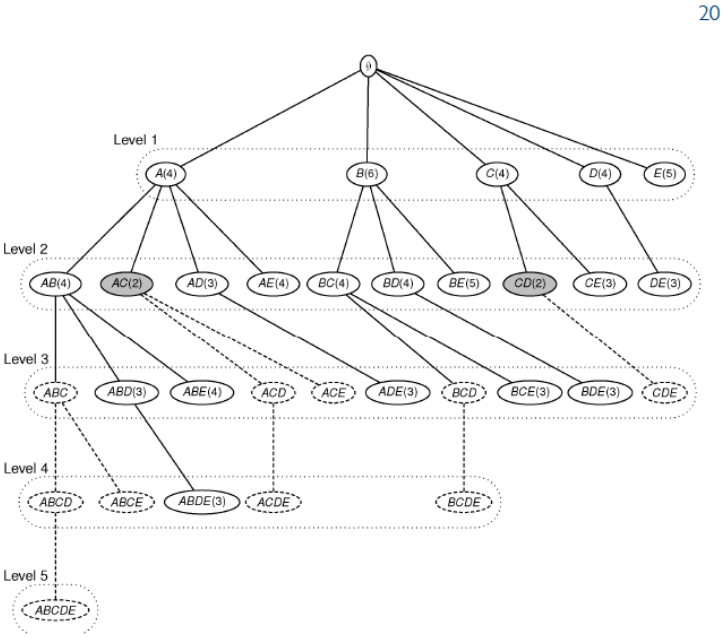
To reduce the complexity :

- Reduce number of candidates **M** by using **pruning** techniques
- Reduce the number of transactions **N** as size of itemset increases
- Reduce number of comparisons **NM** by using efficient data structures and avoiding to match every candidate against every transaction.

4.1.1 Apriori principle

The Apriori principle states that if an itemset is **frequent** than all of its **subsets must also be frequent** (**anti-monotone property of support**)

$$\forall X, Y : (X \subseteq Y) \implies s(X) \geq s(Y)$$



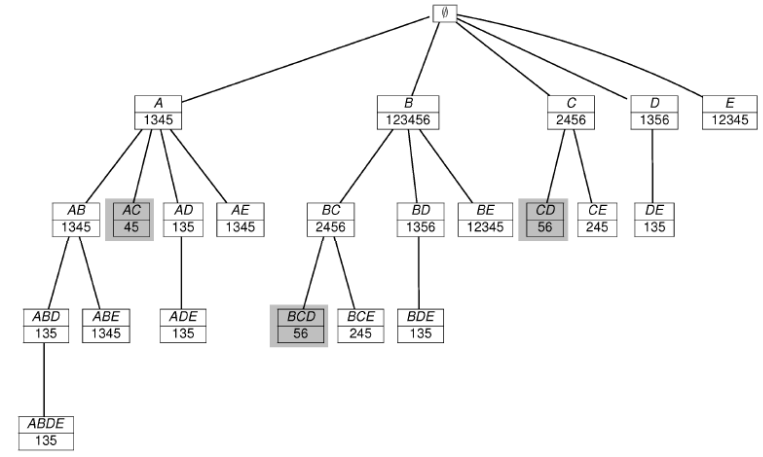
```
APRIORI (D, I, minsup):
1  F ← ∅
2  C(1) ← {∅} // Initial prefix tree with single items
3  foreach i ∈ I do Add i as child of ∅ in C(1) with sup(i) ← 0
4  k ← 1 // k denotes the level
5  while C(k) ≠ ∅ do
6      COMPUTESUPPORT (C(k), D)
7      foreach leaf X ∈ C(k) do
8          if sup(X) ≥ minsup then F ← F ∪ {(X, sup(X))}
9          else remove X from C(k)
10     C(k+1) ← EXTENDPREFIXTREE (C(k))
11     k ← k + 1
12 return F(k)
```

4.1.2 Eclat

Counting during support calculation creates a **bottleneck** which can be solved with this optimized algorithm. The algorithm works like the Apriori but for each itemset the **transaction ids** in which said itemset appears are saved too (**tidset**). The following relationships applies :

$$t(XY) = t(X) \wedge t(Y)$$

$$sup(XY) = |t(XY)|$$

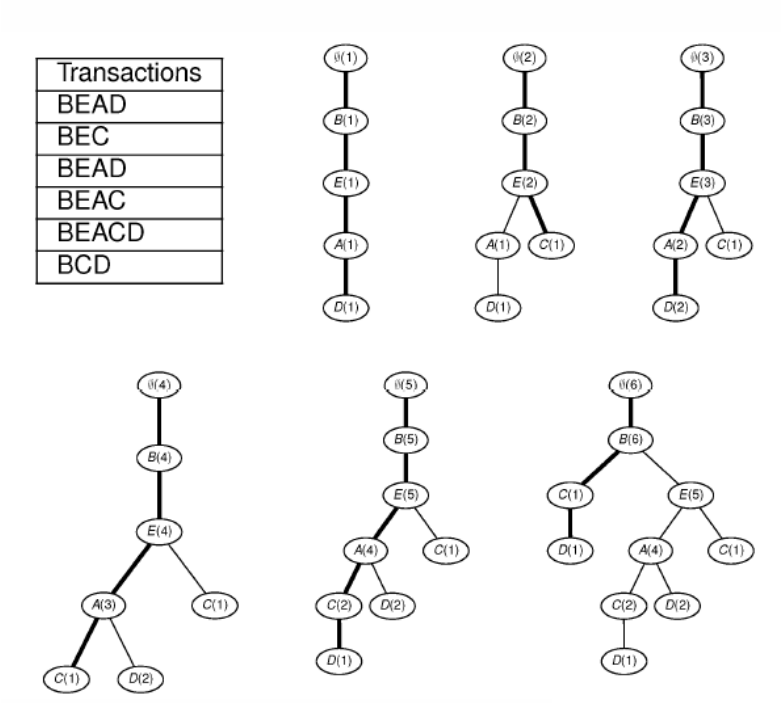


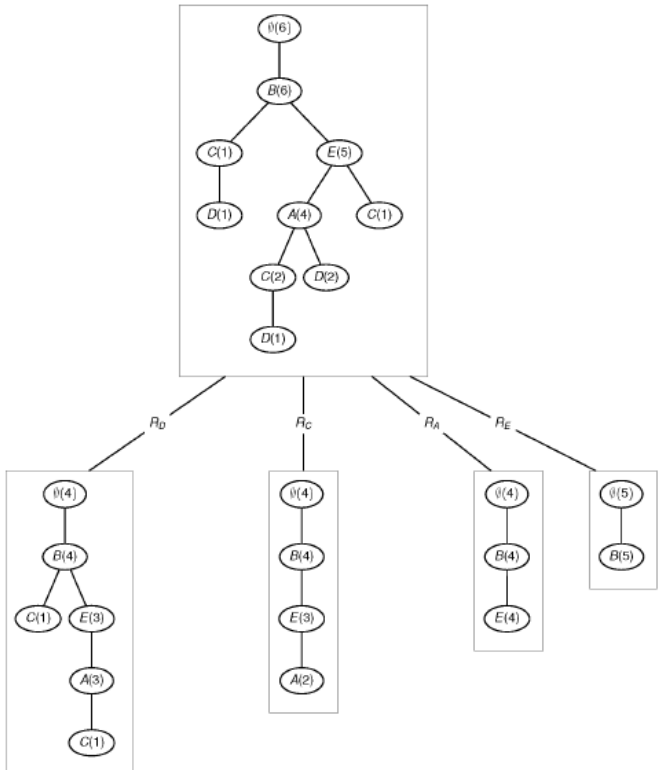
4.1.3 Frequent pattern mining without candidate generation

A big problem with frequent itemset mining is that they can be huge : a 10^4 1-itemset generates 10^7 candidate 2-itemsets. To discover a frequent pattern of size

100 $\{a_1...a_100\}$ about $2^100 \sim 10^30$ candidates must be generated. The problem gets worse considering multiple database scan for support count.
The **Frequent Pattern Tree (FP Tree)** is a perfect solution to **compress** large databases **avoiding costly scans**. The generate-and-test phase from Apriori is completely dropped :

1. Construct FP Tree
2. For each item i compute **projected FP tree**
3. Recursively mine conditional FP Trees and grow frequent patterns so far
4. If conditional FP tree has a **single path** then simply enumerate all the patterns





The method is:

- **Complete**
  It preserves the information for frequent pattern mining and never breaks a long pattern of any transaction

- **Compactness**
  Reduces the relevant information (infrequent items are gone!) without ever exceeding the original database size. Items are in **frequency descending order**.

4.2 Rule generation

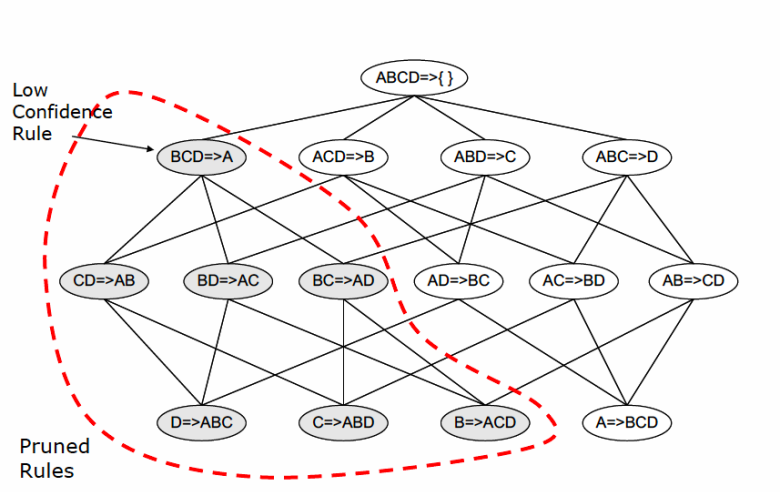
Given a frequent dataset L find all non-empty subsets $f \subset L$ such that $f \implies L-f$ satisfies **the minimum confidence requirement**. If $|L| = k \implies 2^K - 2$ candidate association rules (ignoring $L \to \emptyset$ and $\emptyset \to L$).
Can rules be generated **efficiently** ? Confidence does not follow the **anti-monotone**

property :

$$c(ABC \rightarrow D) \text{ can be larger than } c(AB \rightarrow D)$$

But is follows the anti-monotone property if rule are generated from the **same itemset**:

$$L(\{A, B, C, D\}) : c(ABC \implies D) \geq c(AB \implies CD) \geq c(A \implies BCD)$$



An example with min-conf = 9 :

sup	itemsets
6	B
5	E, BE
4	$A, C, D, AB, AE, BC, BD, ABE$
3	$AD, CE, DE, ABD, ADE, BCE, BDE, ABDE$

1. Start from largest itemset : **ABDE**
2. Generate all subsets A= $\{ABDE, ABD, ABE...\}$
3. Starting from the first subset $X = ABD$ generate rule $ABDE \implies E$ with confidence $\frac{3}{3} > 0.9$.
   Next one is $ABE \implies D$ with confidence $\frac{3}{4} < 0.9$.

4. Each time a subset has not met the required confidence level **all** of its **subsets** are **removed from** `A` (example remove all subsets of ABE from `A`).

5. Finally the algorithm will output all the rules.

```
AssociationRules (F, minconf):
1  foreach Z ∈ F, such that |Z| ≥ 2 do
2      A ← {X | X ⊂ Z, X ≠ ∅}
3      while A ≠ ∅ do
4          X ← maximal element in A
5          A ← A \ X // remove X from A
6          c ← sup(Z)/sup(X)
7          if c ≥ minconf then
8              print X ⟶ Y, sup(Z), c
9          else
10             A ← A \ {W | W ⊂ X}
               // remove all subsets of X from A
```

4.3 Rule assessment measures

If minsup is set **too high** itemsets involving interesting rare items could be missed (like expensive products). On the other hand if it is set **too low** the number of itemset gets very large : a **single** minimum threshold may not be sufficient (applies for confidence too!).

4.3.1 Lift

Lift is the measure of the deviation of **stochastic independence** (=1 then X and Y are independent)

$$\text{lift}(X \implies Y) = \frac{sup(X \cup Y)}{sup(X)sup(Y)} = \frac{conf(X \implies Y)}{sup(Y)}$$

It also measures the **surprise** of the rule : the closer to 1 means that the support of a rule is expected considering the support of its components.
Useful lift values:

- $lift >> 1$: rule **above** expectations
- $lift << 1$: rule **below** expectations

4.3.2 Leverage

Leverage gives an absolute measure of how surprising a rule is and should be used together with lift :

$$\text{leverage}(X \implies Y) = sup(X \cup Y) - sup(X)sup(Y)$$

4.4 Summarizing Itemsets

When dealing with huge databases and itemset is there an efficient way to summarize them? There are some very technical measures :

- **Maximal frequent itemsets**
  A **frequent** itemset is **maximal** has **no frequent supersets** :

  $$M = \{X | X \in F \text{ and } !\exists Y \supset X \text{ such that } Y \in F\}$$

  It gives a **condensed** representation as all subsets of the maximal itemset are **frequent**

- **Closed Frequent itemsets**
  An itemset X is **closed** if all supersets of X have **strictly less support** :

  $$sup(X) > sup(Y), \text{ for all } Y \sup X$$

  The set of **all** closed frequent itemsets is a **condensed** representation as we can determine whether an itemset X is frequent as well as the **exact** support of X using C alone.

- **Minimal generators**
  A frequent itemset X is a minimal generator if it has **no subsets** with the same support

  $$G = \{X | X \in F \text{ and } !\exists Y \subset X, \text{ such that } sup(X) = sup(Y)\}$$

  which means that

  $$sup(X) < sup(Y)$$

4.5 Mining frequent sequences

Until now order was not considered. Sequences are often very important like in market basket analysis to know whether people buy items in sequence o discover navigation patterns on websites.
A **sequence** is an ordered list of symbols : $s = s_1, s_2, ...s_k$ where s_i or $s[i]$ denote the symbol at position i.

$$\text{sequence: } s = s_1, ..., s_n$$

$$\text{sequence: } r = r_1, ..., r_m$$

r is **subsequence** of s ($r \subseteq s$) if there is a function ϕ from $[l, m]$ to $[l, n]$ such that $r[i] = s[\phi(i)]$ and for every position i,j in r : $i < j \implies \phi(i) < \phi(j)$ (order is maintained).
Subsequence r is **consecutive** if $r_1, r_2, ..., r_m = s_j, s_{j+1}, ..., s_{j+m-1}$

- Given s = ACTGAACG
- r_1 = CGAAG is a subsequence of s
- r_2 = CTGA is a consecutive subsequence of s
- r_3 = ACT is a prefix of s
- r_4 = GAACG is a suffix of s

- **Sequence support**
  Given a database **D** containing N sequences , the **support** of a sequence **r** in D is defined as the total number of sequences in D that contains r :

$$sup(r) = |\{s_i \in D | r \subseteq s_i\}|$$

- **Relative support**
  The **relative support** is the percentage of sequences that contain r :

$$rsup(r) = \frac{sup(r)}{N}$$

- **Frequent sequence**
  A sequences is frequent if $sup(r) \geq minsup$

4.5.1 Level-wise mining GSP algorithm

The GSP algorithm work in a similar fashion as the Apriori algorithm . It is a BFS algorithm.

1. Given a set of frequent itemsets at level k, the algorithm generates the candidate for level k+1 and computes the **support** of each candidate and prune the not frequent ones

2. For each sequence s_i in D , check if a candidate **r** sequence is a **subsequence** of s (if yes increment support of r). Then generate all k+1 level candidates

3. For each leaf r_a the sequence is extended with the **last symbol** of any other leaf r_b that shares the **same prefix** (= same parent) so that

$$r_{ab} = r_a + r_b[k]$$

   If r_{ab} is infrequent then prune it.

4.6 Association rule for classification

Given a dataset where classification can be performed (for example data about patient with a class representing if a medicine has worked or not). Association rules assume that the data have a transactional structure.Since classification tasks have data in tabular format , it is possible to find a mapping to perform classification with associations rules:

$$X \implies c_i \quad c_i \text{ is a class label}$$

The confidence , for example , of $X \implies class = yes$ represents the **conditional probability** $P(class = yes|...)$ so the confidence of the association rule is an evaluation of the predictive power (**CBA Algorithm**). It differs from normal association rules because we are working with **equalities** (like $outlook = sunny \implies ...$

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

Here columns can take multiple values , for example $humidity = \{low, medium, high\}$, which cannot be converted into a transactions that are used in association rule mining. A good solution is to use **One hot encoding**:

- $Humidity_low \in \{0, 1\}$
- $Humidity_medium \in \{0, 1\}$
- $Humidity_high \in \{0, 1\}$

5 Regression

Regression can be summarized as

$$\text{Regression} = \text{model building} + \text{model usage}$$

Given N examples , pairs x_i, y_i, **linear regression** computes a model

$$y = w_0 + w_1 x$$

so that for each point

$$y_i = w_0 + w_1 x_i + \epsilon_i$$

The model is evaluated using

$$RSS(w_0, w_1) = \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_i))^2$$

and the goal for linear regression is to **minimize the RSS** by finding the right weights :

$$w_0, w_1 = argmin_{w_0,w_1} RSS(w_0, w_1) = argmin_{w_0,w_1} \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_i))^2$$

To minimize those weights :

1. Set **gradient** to zero $\rightarrow$ **infeasible**
2. Apply **gradient descent**

   $$\text{while not converged } \overrightarrow{w}^{(t+1)} = \overrightarrow{w}^{(t)} - \eta \nabla RSS(\overrightarrow{w}^{(t)})$$

   The **learning rate** η should be set not too large (large steps but may not converge) nor too low (very slow convergence) . So the best is to make it adapt over time

   $$\eta(t) = \frac{\alpha}{t} \text{ or } \eta(t) = \frac{\alpha}{\sqrt{t}}$$

In case of **simple linear regression** there are only two weights (w_0, w_1) so

$$\nabla RSS(\overrightarrow{w}) = \begin{bmatrix} -2 \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_i)) \\ -2 \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_i)) x_i \end{bmatrix}$$

In case of **multiple linear regression** the variables are **more than 1** so there are also **more weights** :

$$y = w_0 + \sum_{j=1}^{D} w_j x_j + \epsilon \quad \text{D input variables}$$

$$RSS(\overrightarrow{w}) = \sum_{i=1}^{N} (y_i - w_0 - \sum_{j=1}^{D} w_j x_{i,j})^2$$

Since the RSS is dependent on the target variable in terms of scale a good **normalized measure** to evaluate a regression algorithm is the R^2 statistic:

$$TSS = \sum_{i=1}^{N} (y_i - \bar{y})^2 \quad \bar{y} = average$$

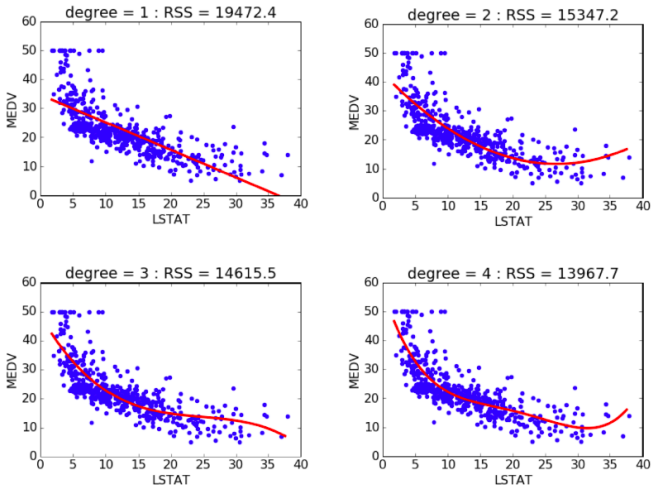
$$R^2 = -\frac{RSS}{TSS}$$

It measures how well the regression line approximates the real data points. When it is 1 it **perfectly fits the data**.
The **general formulation** takes also into account a function $h_j(\overrightarrow{x_i})$, to identify variables **derived** from the original inputs where h_j can be any transformation :

$$y_i = \sum_{j=0}^{D} w_j h_j(\overrightarrow{x_i}) + \epsilon_i$$

$$RSS(\overrightarrow{w}) = \sum_{I=1}^{N} (y_i - \sum_{j=0}^{D} w_j h_j(\overrightarrow{x_i}))^2$$

$$w_j^{t+1} = w_j^{(t)} + 2\eta \sum_{i=1}^{N} h_j(\overrightarrow{x_i})(y_i - \hat{y}_i(\overrightarrow{w})^{(t)})$$



5.1 Model evaluation

Models should be evaluated on data **never seen before** during training : data must be split into **test and training set**.

5.1.1 Hold out evaluation

Reserves a certain amount for testing and the remainder for training:

- **too small** training sets might result in **poor** weight estimation
- **too small** test sets might result in **poor** estimation of future predictions

For small unbalanced datasets,samples might not be representative.Also the selection of the right model degree could depend on the selected test set split.

5.1.2 Cross-validation

1. Split data into k subsets of **equal size**
2. Each subset is in turn used for testing and the remainder for training
3. The error estimates are averaged to yield an overall error estimate

The subsets are often **stratified** to create homogeneous subgroups and reduce the variance.Usually **10-fold CV** is used as extensive experiments have shown that it is the best choice to get an accurate estimate.

5.2 Overfitting

Overfitting means learning the noise in data : good performance is achieved on training sets but bad performance on **test set**! This is often correlated to **large weight estimates** : a good solution is to **penalize** large weights.

5.2.1 Ridge Regression L2 Regularization

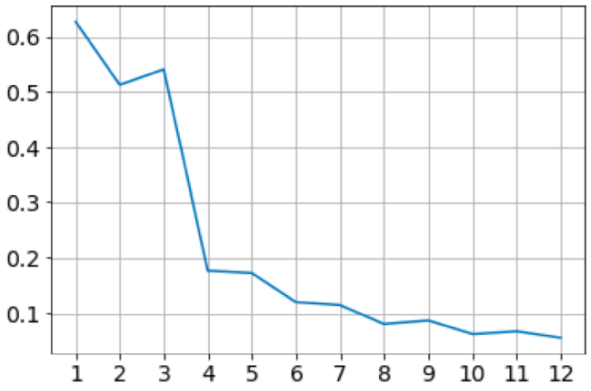
Minimizes the cost function :

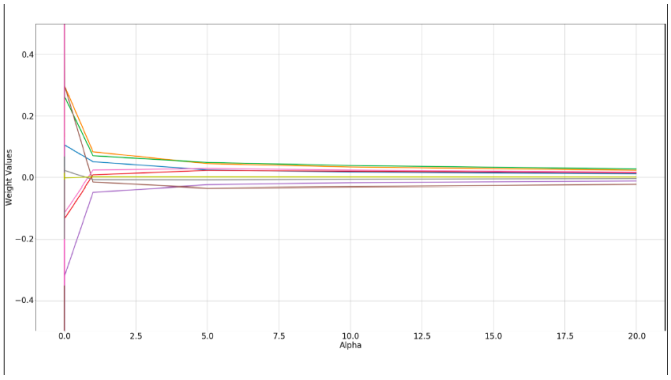
$$Cost(\overrightarrow{w}) = RSS(\overrightarrow{w}) + \alpha||\overrightarrow{w}||_2^2 = \sum_{i=1}^{N}\left(y_i - \sum_{j=0}^{D} w_j h_j(\overrightarrow{w_i})\right)^2 + \alpha\sum_{j=0}^{D} w_j^2$$

- $\alpha = 0$ usual regression
- $\alpha = \infty$ all weights $= 0$

Gradient descent becomes :

$$\Delta w_j = -2\sum_{i=0}^{N} h_j(\overrightarrow{x_i})(y_i - \hat{y}_i(\overrightarrow{w}^{(t)})) + 2\alpha w_j$$





5.2.2 Lasso Regression L1 Regularization

Minimizes the cost function :

$$Cost(\overrightarrow{w}) = RSS(\overrightarrow{w}) + \alpha||\overrightarrow{w}||_1 = \sum_{i=1}^{N}\left(y_i - \sum_{j=0}^{D} w_j h_j(\overrightarrow{w_i})\right)^2 + \alpha\sum_{j=0}^{D}|w_j|$$

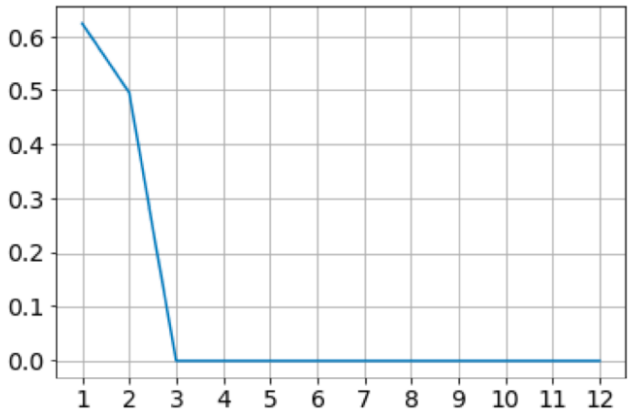
- $\alpha = 0$ usual regression
- $\alpha = \infty$ all weights $= 0$

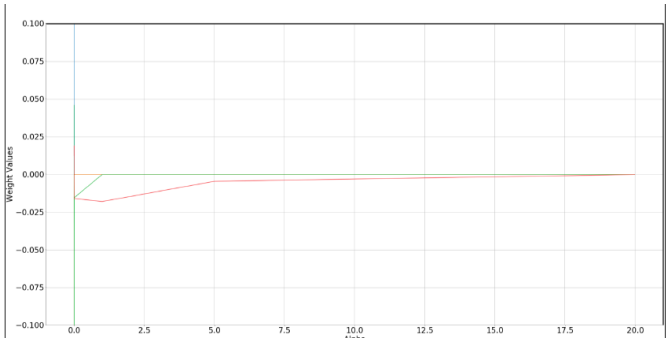
Gradient descent becomes :

$$z_j = \sum_{i=1}^{N} h_j(\vec{x_i})^2$$

$$\rho_j = \sum_{i=1}^{N} h_j(\vec{x_i})(y_i - \hat{y}_i(\vec{w}_{-j}^{(t)}))$$

$$w_j = \begin{cases} (\rho_j + \alpha/2)/z_j & \texttt{if } \rho_j < -\alpha/2 \\ 0 & \texttt{if } \rho_j \in [-\alpha/2, \alpha/2] \\ (\rho_j - \alpha/2)/z_j & \texttt{if } \rho_j > \alpha/2 \end{cases}$$





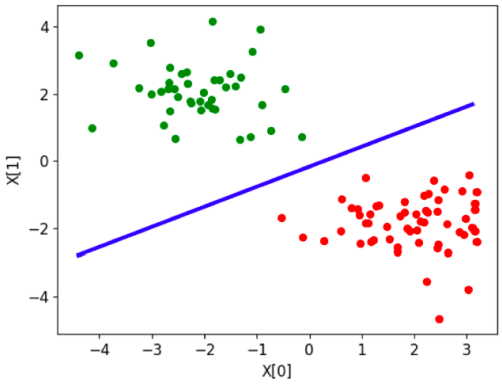
Lasso tends to **zero out less important feature** applying a so called **feature selection** which produces **sparser solutions**

5.2.3 Alpha selection

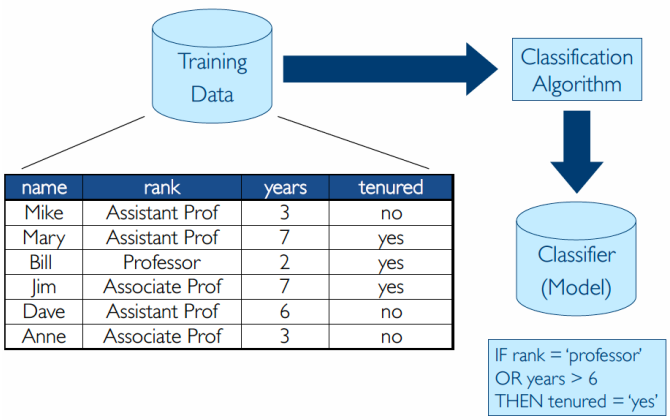
The values of alpha should be selected using a **dev set** or **validation set** : a third set, different from test and training ,used to selected the right penalization parameter. The test set cannot be used since it using the selected alpha. The best solution is to split again the training set and obtain the validation set (if there is **enough data**).
If there is not enough data , α can be selected by applying k-fold cross-validation over the training data choosing α corresponding to the lowest average cost over k-fields.

6 Classification

In classification data have been labeled according to a certain **concept** (good vs bad, positive vs negative etc...) identified by a **supervisor** who labeled the data. The goal of classification is to compute a model using known that can **discriminate** between examples that have been labeled differently



The model is then used to **label unseen point** in the prediction phase.Differently form regression where the target variable is **continuous** , in classification the target variable is **discrete**.
An example using **classification rules**:



name	rank	years	tenured
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

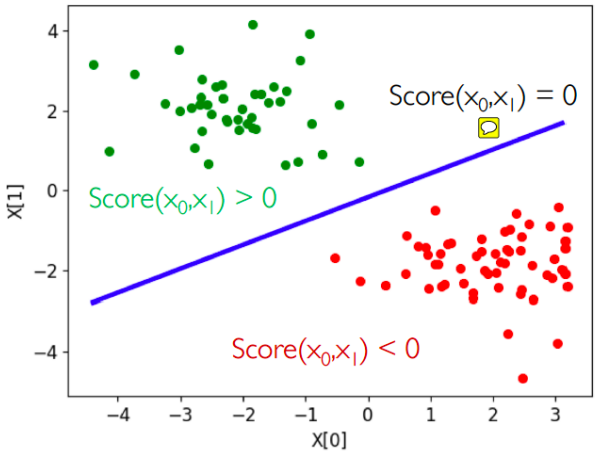
6.1 Model evaluation

- **Accuracy**

- **Speed** (of fitting and prediction)
- **Robustness,scalability...**

6.2 Logistic Regression

The most simple classification algorithm. Starting from the basic idea behind classification algorithms :



$$Score(\overrightarrow{x_i}) = \sum_{j=0}^{D} w_j h_j(\overrightarrow{x_i})$$

The label is determined as follows:

$$\hat{y} = \text{sign}(Score(\overrightarrow{x_i})) = \begin{cases} +1 & \text{if } Score(\overrightarrow{x_i}) \geq 0 \\ -1 & \text{if } Score(\overrightarrow{x_i}) < 0 \end{cases}$$

Logistic regression behaves similarly but instead of computing a label the algorithm computes the **probability of assigning a class to an example**:

$$P(y_i|\overrightarrow{x_i})$$

For example for the positive class +1 :

$$P(\hat{y}_i = +1|\overrightarrow{x_i}) = \frac{1}{1 + e^{-Score(\overrightarrow{x_i})}}$$

Now adding explicitly feature transformations and weights:

$$P(\hat{y}_i = +1|\overrightarrow{x_i}, \overrightarrow{w}) = \frac{1}{1 + e^{-\overrightarrow{w}h(\overrightarrow{x_i})}}$$

The weights are computed using **maximum likelihood** :

$$l(\overrightarrow{w}) = \prod_{i=0}^{N} P(y_i|\overrightarrow{x_i}, \overrightarrow{w})$$

This is maximized using **gradient descent** using the log-likelihood:

$$ll(\overrightarrow{w}) = ln(\overrightarrow{w})$$

which updates weights j using :

$$\frac{\partial ll(\overrightarrow{w})}{\partial w_j} = \sum_{I=1}^{N} h_j(\overrightarrow{x_i}) \left(1[y_i = +1] - P(y = +1|\overrightarrow{x_i}, \overrightarrow{w})\right)$$

So

$$\hat{y} = \begin{cases} +1 & \sum_{j=0}^{D} w_j h_j(x) \geq 0 \\ -1 & \sum_{j=0}^{D} w_j h_j(x) < 0 \end{cases}$$

6.3 Overfitting and regularization

Like regression :

- **L1 Regularization** : $l(\overrightarrow{w}) - \alpha||\overrightarrow{w}||_1$
- **L2 Regularization** : $l(\overrightarrow{w}) - \alpha||\overrightarrow{w}||_2^2$

6.4 Multiclass Classification

Logistic regression assumes that the output classes are just **two**. If there are more than 2 target classes, multiclass classification must be used. A possibility is to use the **One vs the rest** method :

- For each class it creates **one classifier** that predicts the target class against all others
- Then given an example **all classifiers** are used and the label with the **highest probability** is returned

In alternative a **multinomial multiclass** model can be used.

6.5 Categorical values

If the variables are **categorical** instead of numerical a good solution is to use **One Hot Encoding** which maps categorical values into **binary 0/1** values , each one describing a specific attribute. For example the feature `Outlook` can hold three values { Sunny,Rainy,Overcast} :

- Outlook-sunny $\in \{0, 1\}$
- Outlook-rainy $\in \{0, 1\}$
- Outlook-overcast $\in \{0, 1\}$

7 Credibility : robust evaluation methods

When creating a model it is important to know :

1. how the model **performs**
2. how **reliable** the performance estimates are
3. how to **compare** performance of competing models
4. which model to choose if dealing with two equally performing models

7.1 Metric for performance evaluation

7.1.1 Regression

- **Residual sum of squares**

$$RSS(\overrightarrow{w}) = \sum_{i=1}^{N} \left(y_i - \sum_{j=0}^{D} w_j h_j(\overrightarrow{x_i}) \right)^2$$

- **Coefficient of determination**

$$TSS = \sum_{i=1}^{N} (y_i - \bar{y})^2$$

$$R^2 = 1 - \frac{RSS}{TSS}$$

$$R_2^2 = \frac{\sum_{i=1}^{N} (y_i \hat{-} \bar{y})^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}$$

- **Mean square error**

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

- **Root mean square error**

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2}$$

7.1.2 Classification

The **Confusion matrix** is a good way to evaluate classification tasks:

		PREDICTED CLASS	
		Yes	No
TRUE CLASS	Yes	a (TP)	b (FN)
	No	c (FP)	d (TN)

a: TP (true positive) c: FP (false positive)

b: FN (false negative) d: TN (true negative)

- **Accuracy**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

How many right predictions over all the outcomes. **Not always a good choice** : for example if 990/1000 example are class 0 and 10/1000 examples are class 1 predicting always class zero gives accuracy 99.9% which is useless since the interest is to predict class 1 (bank fraud example).

- **Cost**

$$C(x) = \text{ cost of missclassifying example of type x}$$

For example **FN** can be more "**expensive**" than **FP**:

Cost Matrix		PREDICTED CLASS	
	C(.)	+	-
ACTUAL CLASS	+	-1	100
	-	1	0

Model M_1		PREDICTED CLASS	
		+	-
ACTUAL CLASS	+	150	40
	-	60	250

Accuracy = 80%

Cost = 3910

Model M_2		PREDICTED CLASS	
		+	-
ACTUAL CLASS	+	250	45
	-	5	200

Accuracy = 90%

Cost = 4255

This cost matrix can be used by some algorithms (like **Decision Trees**) to guide them.

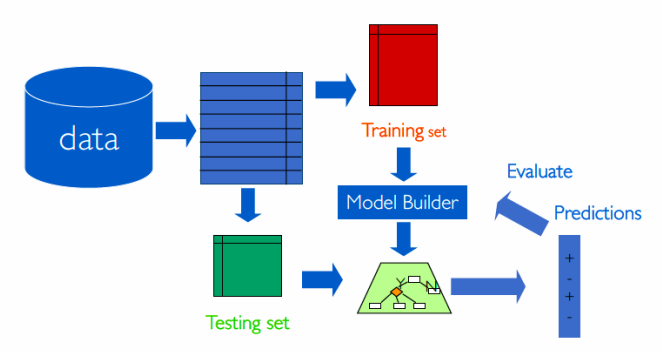
- **Precision**
  Focuses on the percentage of examples that have been classified **positive** and **are actually positive**
  $$\frac{TP}{TP + FP}$$
  higher precision = lower FP

- **Recall**
  Focuses on the percentage of examples that have been classified as positive with respect of the number of good existing examples
  $$\frac{TP}{TP + FN}$$
  higher recall = lower FN

- **F1 measure**
  Is biased towards all except **TN**
  $$\frac{2\text{Recall} * \text{Precision}}{\text{Precision} + \text{Recall}}$$

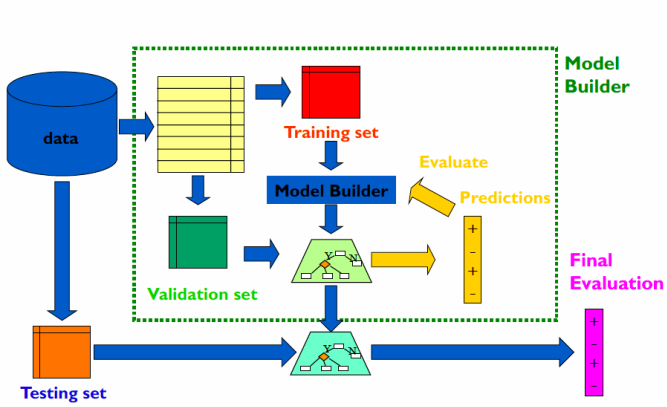
7.2 Methods for performance evaluation

As already seen for regression and classification different methods can be used to evaluate a model (train/test split, CV...).



It is important that **test data** is **never** used to train a model nor used to **tune parameters** :

- Train
- Test
- Validation (used to tune parameters)



Performance of a model can depend on many factor (class distribution , cost of missclassfication , size of training and test sets) and the estimated performance depends on the many methods :

- **Holdout**
  Reserve $\frac{1}{3}$ $\frac{2}{3}$ for training and $\frac{2}{3}$ $\frac{1}{3}$ for testing. This method is **bad** for **small, unbalanced** as samples might not be representative (it can produces training or test sets with only 1 class!).For this reason **stratified** sampling should be used : it makes sure that each class is represented **equally** in both subsets.
- **Repeated holdout**
  Holdout estimate can be made more reliable by **repeating** the process with **different** subsamples.In each iteration a certain proportion is **randomly selected** for training. The error rates are then **averaged** to yield an overall error rate.
  The bad thing is that different test sets **overlap**
- **Cross validation**

1. Split data in k subsets of equal size

2. Each subset used in turn for testing while the rest for training.

Better than repeated holdout because it does not overlap test sets. Even better performances are obtained by **shuffling** and **stratifying** (which reduces variance). Errors are again **averaged**.

- **Leave-One-Out CV**
  It is a particular form of CV that sets the number of fold = number of instances of training set. This makes best use of the data as only **one sample** is used for testing and the rest for training without using random sampling. The downside is that it is **computationally expensive** and does not allow **stratification**.

- **Bootstraping**
  Cross validation uses sampling without **replacement** : the same instance, once selected , can not be selected again for a particular training/test set. Bootstrap allows to **replace** samples :

  - Sample a dataset of n instances with replacement to form a **new dataset** with n instances.
  - Use this dataset as **training set**
  - Use the instances that **don't occur in new training set** as **test set** (taken from original train set) An instance has probability $1 - \frac{1}{n}$ of **not** being picked , so ending up in the **test data** with probability :

    $$\left(1 - \frac{1}{N}\right)^N \approx e^{-1} = 0.368$$

    This means that the training data contain approximately 63.2% of the instances.
    The **error estimate** will be very **pessimistic** since only 63.2% of the data has been used. A good solution is to compute the **overall error** of training and testing :

    $$\epsilon = 0.632 \cdot \epsilon_{test} + 0.368 \cdot \epsilon_{train}$$

    . This process can be **repeated** with different replacement samples, averaging out the results.

7.3 Compare competing models

Suppose there are two models :

- Model M_A with accuracy 82% using 10-fold CV
- Model M_B with accuracy 80% using 10-fold CV

How much **confidence** can we place on accuracy M_A and M_B? Is the statement M_A is better than M_B correct or can the difference be explained as "random fluctuations" in train/test set or even random luck? To answer to this questions is to **measure the odds**:

- Apply **t-test**
- Compute **p-value** (probability that the reported difference is due to chance)

An overall view of the possible combinations :

	Nonparametric tests		**Parametric tests**
	Nominal data	Ordinal data	Ordinal, interval, ratio data
One group	Chi square goodness of fit	Wilcoxon signed rank test	One group t-test
Two unrelated groups	Chi square	Wilcoxon rank sum test, Mann-Whitney test	Student's t-test
Two related groups	McNemar's test	Wilcoxon signed rank test	Paired Student's t-test
K-unrelated groups	Chi square test	Kruskal -Wallis one-way analysis of variance	ANOVA
K-related groups		Friedman matched samples	ANOVA with repeated measurements

7.3.1 Paired t-test using CV

Generate **k-folds** and for each configuration compute the performances of Model A and Model B :

- $\theta_1^A ... \theta_k^A$
- $\theta_1^B ... \theta_k^B$

- $\delta_i = \theta_i^A - \theta_i^B$
- $\mu_\delta = \frac{1}{k} \sum_i \delta_i$ (mean)
- $\sigma_\delta = \sqrt{\frac{1}{k} \sum_i (\delta_i - \mu_\delta)^2}$ (stdev)

The define the **two hypothesis**:

- $H_0 : \mu_\delta = 0$
- $H_1 : \mu_\delta \neq 0$

Then the **t-test** is applied to check whether the **null hypothesis** H_0 can be rejected with a **target confidence** :

ALGORITHM 22.4. Paired *t*-Test via Cross-Validation

```
PAIRED t-TEST(α, K, D):
D ← randomly shuffle D
{D_1, D_2, ..., D_K} ← partition D in K equal parts
foreach i ∈ [1, K] do
    M_i^A, M_i^B ← train the two different classifiers on D \ D_i
    θ_i^A, θ_i^B ← assess M_i^A and M_i^B on D_i
    δ_i = θ_i^A − θ_i^B
μ̂_δ = (1/K) Σ_{i=1}^K δ_i
σ̂_δ^2 = (1/K) Σ_{i=1}^K (δ_i − μ̂_δ)^2
Z_δ^* = √K μ̂_δ / σ̂_δ
if Z_δ^* ∈ (−t_{α/2,K−1}, t_{α/2,K−1}) then
    Accept H_0; both classifiers have similar performance
else
    Reject H_0; classifiers have significantly different performance
```

For example fixing the confidence at 95% then the reported difference is statistically significant if the **p-value** is smaller than $1 - c = 0.05$, otherwise if the **p-value** is **larger** than 0.05 then the difference is **not** significant.
Note that the test can be done **paired** (estimates from **same** dataset) or **unpaired** (estimate from **differnet** datasets).

7.3.2 Multiple testing

Comparing performance of **several** classification algorithms requires a different approach . This is because :

- P(making a mistake) = 0.05

- P(not making a mistake) = 0.95
- P(making a mistake)^{20} = 0.358
- P(not making a mistake)^{20} = 0.642

So when using 20 different methods there is a 64.2% chance of making at least one mistake.
Solutions:

- **Bonferroni Test**
  Assumes that individual tests are independent.
  Divides the threshold by the number of tests performed and **still use the t-test**:
  - $\frac{0.05}{20} = 0.0025$
  - P(making a mistake) = 0.0025
  - P(making a mistake)^{20} = 0.0488
- **Non-Parametric tests**
  A better way is to use non-parametric tests : they do not make any assumptions about the **distribution** of the variable population.
  The two main tests are :
  - **Mann-Whitney U Test** : non parametric equivalent of t-test
  - **Wilcoxon matched pairs singed rank test** : used to compare two related groups

7.3.3 Probabilistic classifiers

As already seen Logistic Regression is a classification method that used **probability** to predict labels

$$P(y_i|\overrightarrow{x_i})$$

So given an example x_i it predicts a label with the **largest probability**, which is the equivalent of using a **threshold** of **0.5** :

$$P(+1|x_i) = 0.55$$

$$P(-1|x_i) = 0.45$$

Clearly the chosen class is $+1$ as it is ≥ 0.5

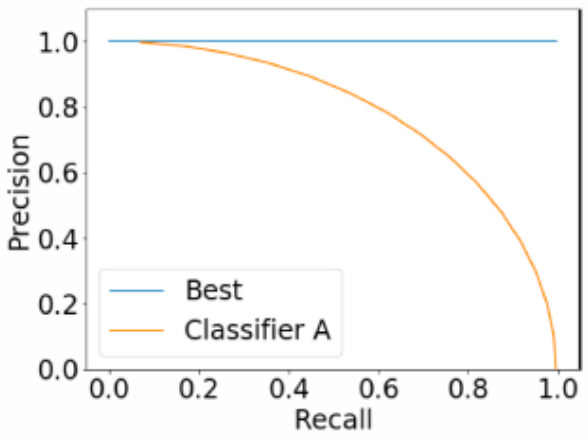
But **different thresholds** can be used : for example when we need a **bigger confidence** that the assigned label is 1 a threshold of 0.75 can be used.
Selecting a threshold **close to 1** (**pessimistic classifier**):

- **High precision** : not likely to produce **false positive** because the given threshold allows only very confident predictions to assign label 1.
- **Low Recall**: there are likely many **false negatives**

Selecting a threshold **close to 0** (**optimistic classifier**):

- **Low Precision** : almost everything is positive
- **High Recall** : minimum number of **false negatives**

Trade-off to **optimize** threshold!
Precision-Recall curves :



Different classifiers show different shapes, to decide which is the best one:

- **Area under the curve** (closer to 1 = better)
- **F1- Measure**

7.3.4 ROC Curve

Receiver operating characteristic curve plots the **True positive** rate against the **False positive rate**

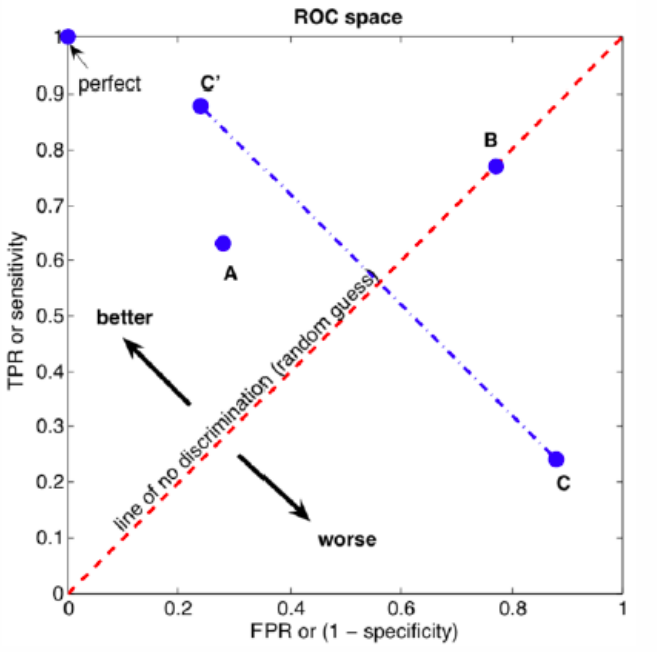
$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Each performance of a **classifier** is a **point** on the ROC curve. The location on the ROC curve of a classifier depends on :

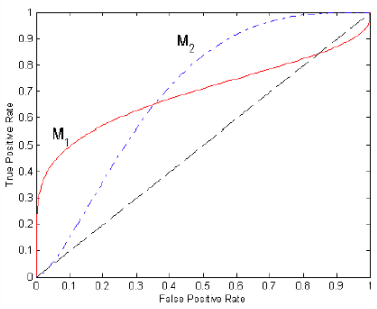
- Threshold used for classification
- Sample distribution
- Cost matrix changes

The ideal situation is point **(0,1)** $\rightarrow$ **no** false positive , only **true positive**.



- No model consistently outperform the other
- M_1 is better for small FPR
- M_2 is better for large FPR

- Area Under the ROC curve
- Ideal, area = 1
- Random guess, area = 0.5



7.4 Model Selection

When building a model the trade-off between **model complexity** and **prediction accuracy** on training data is important : a good model must be **as simple** as possible but achieve high accuracy on the training data. But finding the **best** algorithm is not always easy , as state by the **No Free Lunch Theorem**: if one algorithm **outperforms** another in a certain situation than it is due of its **fit to the particular problem** rather than the **superiority of the algorithm**. When dealing with a new problem it is important to gather information about **priori knowledge, data distribution,amount of data and cost/rewards.**

8 Decision Trees

Decision Tree's is an algorithm that has:

- **Internal Nodes**: which perform **tests** on **attributes**
- **Branches**: that represent **outcomes** of the test
- **Leaf Nodes**: represents a **class label** (or class label distribution)

At each node of the DT one attribute is chosen to **split** training samples into **distinct classes** as much as possible. After the model is computed , a new case is identified by following a **matching path** to a **leaf node**.

8.1 Computing Decision Trees

- **Top-down construction**
  All training samples are at the root, then recursively partitioned by choosing one attribute at a time.
- **Bottom-up Pruning**
  Remove sub-trees or branches, in a bottom up manner to **improve** the estimated accuracy on new cases.

Decision Trees should separate classes in a **pure** way , that means that the separated areas contain mainly examples of one class. On this basis , DT implement a **purity** or **impurity** measure :

- ID3 $\rightarrow$ **information gain**
- C4.5 $\rightarrow$ **information gain ratio**
- CART $\rightarrow$ **gini index**

8.1.1 Information Gain - ID3

Information gain increases with the **average purity** of the subsets that an attribute produces . The **splitting strategy** chooses the attribute that results in the **largest information gain**.

Information is computed in **bits** : given a probability distribution the info required to predict an event is the **distribution's entropy** (which gives the information required in bits).

$$entropy(p_1, ..., p_n) = -p_1 logp_1 - p_2 logp_2 ... - p_n logp_n$$

So for each attribute the DT applies the **entropy measure** :

- outlook = sunny $\rightarrow entropy(\frac{2}{5}, \frac{3}{5}) = .971$
- outlook = overcast $\rightarrow entropy(\frac{5}{5}, \frac{0}{5}) = .000$
- outlook = rainy $\rightarrow entropy(\frac{3}{5}, \frac{2}{5}) = .971$
- $info([2,3],[4,0],[3,2]) \rightarrow \frac{5}{14} \cdot 0.971 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971$

After computing the information , the **gain of information** (before and after split) must be computed :

$$gain(A) = info(D) - info_A(D)$$

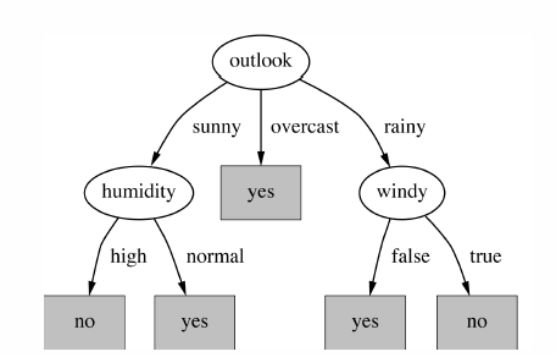
where

$$info(D) = -p_1 logp_1 - ... - p_n logp_n$$
$$info_A(D) = \frac{|D_1|}{D} info(D_1) + ... + \frac{|D_n|}{|D|} info(D_n)$$

In our example :

- gain(outlook) $= info([9,5]) - info([2,3],[4,0],[3,2]) = .940 - .693 = 0.247$

The higher the gain, the better! The final tree looks like this:



Splitting **stops**:

- All samples given a node belong to the same class
- No further attributes available → majority voting
- No samples left
- No gain in splitting

Using decision trees 100% accuracy can be achieved but it is **not** a desirable thing to have most of the time. For example an **ID** attribute, where each sample has a distinct value , will surely be picked by the algorithm as it results in the highest gain. But an ID will never be a good prediction indicator so will lead to **overfitting**.
This is why Information gain is **biased** towards choosing attributes with a **large number of values**.

8.1.2 Information Gain Ration - C4.5

Modification of the Information Gain that **reduces bias towards highly-branched** attributes. Information gain should be:

- **Large** → data **evenly spread**
- **Small** → data belong to **one branch**

Information gain ratio takes **number** and **size** into account when choosing an attribute. The **intrinsic information** computes the entropy of distribution of instances into branches:

$$IntrinsicInfo(S, A) = -\sum \frac{|S_i|}{|S|} log \frac{|S_i|}{S}$$

$$GainRatio(S, A) = \frac{Gain(S, A)}{IntrinsicInfo(S, A)}$$

So for example for an attribute like ID :

- Intrinsic Information → $info([1, 1, ..., 1]) = 14 \cdot (-\frac{1}{14} \cdot log\frac{1}{14}) = 3.807$
- GainRatio(ID-Code) = $\frac{.940}{3.807} = 0.246$

C4.5 works also with **numerical attributes** :

1. sort all values and their class labels

2. check **all cutting points** and choose the one with the best information gain

Sorting instances by value takes $O(nlogn)$ but has to be done **once** since the ordering for children can be **derived** from the parents with derivation time $O(n)$ which requires to **store indices** for each numeric attribute.
Another feature of numeric attributes is that they can be used more than ones in different part of the tree. This can lead to very complex trees and is often a problem. Solutions:

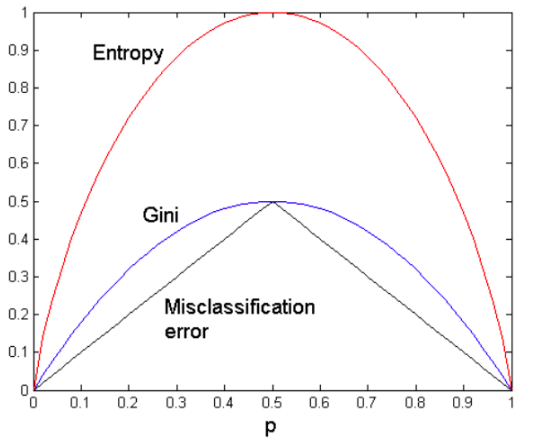
- **Pre-discretize** numeric attributes
- Use **multi-way splits** instead of binary splits.

8.1.3 Gini Index - CART

The Gini Index ,for a dataset T contains examples from n classes, is defined as

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

Where p_j is the **relative frequency** of class j in T. The index is **minimized** if the classes in T are **skewed**.



Gini and entropy measure the same thing but have different shapes in relation to the frequency of a class.

If the dataset is split into two **subsets** by attribute A:

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

With a reduction of **impurity**:

$$\Delta gini(A) = gini(D) - gini_A(D)$$

The attribute that provides the **smallest** Gini splitting D over A (or the **largest reduction in impurity**) is chosen to split the node. This requires to enumerate all the possible splitting points for each attribute. Example over the `Outlook` attribute :

- 9 tuples labeled yes , 5 labeled no :

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- The attribute has **three** possible values so evaluate all possible partitions:
  - (overcast,rainy) , (sunny)

$$Gini(D_{o,r}, D_s) = \frac{9}{14} Gini(D_{o,r}) + \frac{5}{14} Gini(D_s)$$

$$= \frac{9}{14} Gini([7,2]) + \frac{5}{14} Gini([2,3]) = 0.394$$

  - (rain,sunny) , (overcast)

$$Gini(D_{r,s}, D_o) = \frac{10}{14} Gini(D_{r,s}) + \frac{4}{14} Gini(D_o)$$

$$= \frac{10}{14} Gini([5,5]) + \frac{4}{14} Gini([4,0]) = 0.357$$

  - (overcast,sunny) , (rainy)

$$Gini(D_{o,s}, D_r) = \frac{6}{2} Gini(D_{s,o}) + \frac{3}{2} Gini(D_r)$$

$$= \frac{6}{3} Gini([6,3]) + \frac{3}{2} Gini([3,2]) = 0.457$$

- The best partition is (rainy,sunny) , (overcast) with a gain of

$$Gini(D) - Gini(D_{r,s}, D_o) = 0.459 - 0.357 = .102$$

8.2 Overfitting and generalization

A decision tree with too many branches may reflect **anomalies** or **outliers** : results in poor accuracy for unseen examples. Solutions :

- Pre-pruning
- Post-pruning

8.2.1 Pre-pruning

Pre-pruning bases its pruning technique on **statistical significance tests** : only statistical significant attributes were allowed to be selected by information gain procedure. The tree stops growing when there is no statistical significant **association** between any attribute and the class at a particular node.
The most popular test is the **Chi-Squared test** already used in the ID3 algorithm.

8.2.2 Post-pruning

Removes branches from **full-grown** trees applying :

- **Subtree raising**
- **Subtree replacement**
  Works bottom-up: it considers replacing a tree if and only if all its subtrees have been considered. This is done using again statistical tests.

Using a strategy :

- **Error estimation**
  A branch/tree is pruned only if it reduces the **estimated error**. Error on training data is **not** a useful measure : keep an **hold-out set** for pruning (**reduced error pruning**).
  C4.5 derives a **confidence interval** on the training data and uses a heuristic limit for pruning (Standard Bernoulli based method):

$$e = \frac{\left(f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}\right)}{\left(1 + \frac{z^2}{N}\right)}$$

If $c = 25\% \rightarrow z = 0.69$, f is the error on training data and N the number of instances covered by the leaf.

- **Significance testing**
- **MDL principle**

8.3 Regression and model trees

DT can also be used to perform regression working in the same way as classification trees : looking for the best split that minimizes the impurity measure and the standard deviation in the leaves is very small

$$\text{impurity measure : SDR} = \sigma(D) - \sum \frac{|D_i|}{|D|}\sigma(D_i)$$

- D = original dataset
- D_i= partitions
- σ= standard deviation of the target attribute in the set.

Prediction is computed as the **average** of numerical target variables in the subspace (**regression trees**) or leaf nodes contain linear models to predict the target value (**model trees**)

8.3.1 Decision Stumps

Decision stumps are **one level decision trees** . They are very simple and also the main building block for **boosting methods**.

- **Categorical values**
  One branch for each attribute, one branch for one value and one for the others; missing values a special value
- **Numerical values**
  Two leaves defined by a threshold value selected based on some criterion ; multiple splits

9 Classification Rules

Given a dataset

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

A set of **rules** can be used to classify data:

- IF (humidity = high) AND (outlook = sunny) THEN play=no (3.0/0.0)
- IF (outlook = rainy) AND (windy = TRUE) THEN play = no (2.0/0.0)
- OTHERWISE play = yes (9.0/0.0)

A classification rule are **IF-THEN rules** where the IF part states conditions (**propositional with comparison and FOL Horn clauses**) over data and the THEN part includes a class label.
A classification rule is characterized by :

- **ncovers** = number of examples covered by the rule
- **ncorrect** = number of examples correctly classified by the rule
- **coverage(R)** = $\frac{\text{ncovers}}{\text{size of training data set}}$
- **accuracy(R)** = $\frac{\text{ncovers}}{\text{ncorrect}}$

If more than 1 rule is triggered a **conflict** can happen and must be solved:

- **Size ordering** = assign highest priority to the triggering rule that has the **toughest** requirements (= with the most attribute test)
- **Class-based ordering** = decreasing order of **prevalence** or **missclassification cost** per class

- **Rule-based ordering** = rules are organized into a long priority list according to some measure of rule quality

Rule learning can be done in two ways:

- **Direct Method** = directly learn the rules from the training data
- **Indirect Methods** = learn decision trees or NN and from there convert/extract rules

9.1 Direct Method

9.1.1 IR Classifier

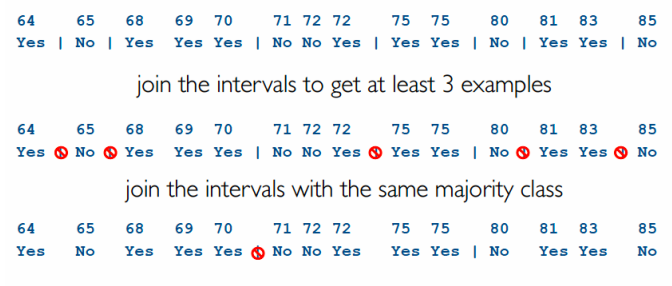
IR Classifier learns a simple rule involving **one attribute** (assumes **nominal** attributes) in a similar way to a decision stump. The rule tests **all values** of one particular attribute.

1. One branch for each value
2. Each branch assigns most frequent class
3. Performance measured using **error rate** computed as the proportion of instances that don't belong to the majority class of their corresponding branch. Missing value = separate value.
4. Choose attribute with lowest error rate

Numerical attributes can be used but must be **discretized** : decreases complexity but at the cost of losing some information. IR applies **supervised discretization** :

- Sort instances according to **attribute's value**
- Place **breakpoints** where class changes (majority class)

This procedure is **very sensitive to noise** and can lead to **overfitting** . To avoid this a **minimum number** of instances can be enforced:



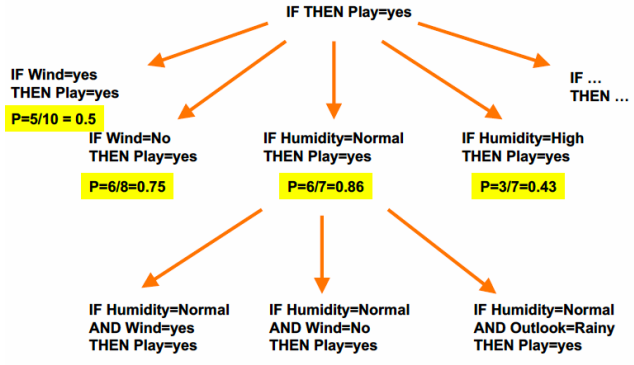
9.1.2 Sequential Covering

1. Take all data and generate one rule
2. **Eliminate** all **positive** examples **covered** by the rule
3. repeat till all examples are covered

```
procedure Covering (Examples, Classifier)
input: a set of positive and negative examples for class c

// rule set is initially empty
classifier = {}

while PositiveExamples(Examples)!={}
  // find the best rule possible
  Rule = FindBestRule(Examples)
  // check if we need more rules
  if Stop(Examples, Rule, Classifier) breakwhile
  // remove covered examples and update the model
  Examples = Examples\Cover(Rule,Examples)
  Classifier = Classifier U {Rule}
Endwhile
// post-process the rules (sort them, simplify them, etc.)
Classifier = PostProcessing(Classifier)
output: Classifier
```



This principle is used for example in **PRISM** :

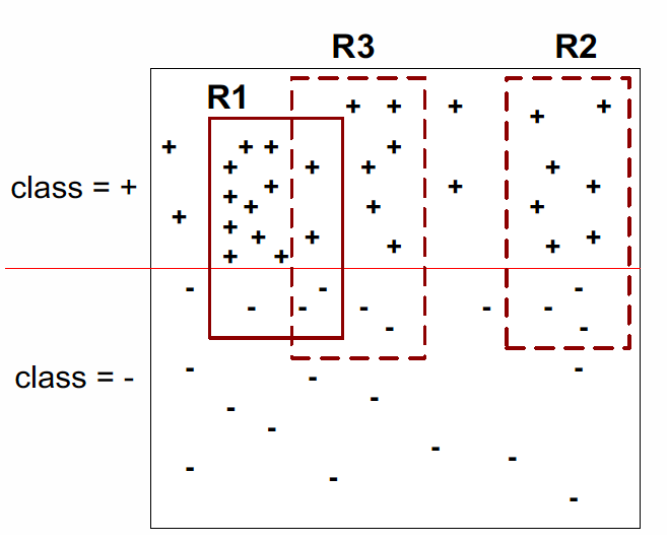
```
For each class C
  Initialize E to the instance set
  While E contains instances in class C
    Create a rule R with an empty left-hand side that predicts class C
    Until R is perfect (or there are no more attributes to use) do
      For each attribute A not mentioned in R, and each value v,
        Consider adding the condition A = v to the left-hand side of R
        Select A and v to maximize the accuracy p/t
          (break ties by choosing the condition with the largest p)
      Add A = v to R
    Remove the instances covered by R from E
```

Different **rule testing mechanisms** can be used:

- **Accuracy** $\rightarrow \frac{p}{t}$ total over positive, produces rules that cover only positive instances as quick as possible but may produce rule with **small coverage** (noise?)
- **Information gain** $\rightarrow p(log\frac{p}{t} - log\frac{P}{T})$ where P , T are positive and total **before** the rule was added.

Eliminating covered instances is **a must** to avoid generating identical rules:

- Eliminating **positive** instances ensures that rules are different and that the accuracy is not **overestimated** which results in a **robuster** accuracy
- Eliminating **negative** instances prevents **underestimating** the rule's accuracy



In this case Rule 3 has :

- **true score** $\frac{8}{12} = .66$
- eliminating positive $\frac{6}{10} = .6 < .66 \rightarrow$ **prevent overestimating**
- eliminating negative $\frac{8}{10} = .8 > .66 \rightarrow$ **prevent underestimating**

Missing values and numeric attributes

Missing values usually **fail the test** and must be handled in a special way (for example considering missing as a **special value**). Otherwise the covering algorithm can

- leave them uncovered until later
- use other test to test positive examples

Numeric attributes are treated just like in DT , with **binary split points** which can be found by optimizing test selection criterion

Stopping Criterion and Rule Pruning

The process usually stops when there is **no** significant improvement by adding a rule (similar to post-pruning in DT).
Reduced error pruning :

1. Remove **one conjunct** from a rule
2. Compare error rate on validation set
3. If error improves ,**prune conjunct**

Pruning can be done using **incremental pruning** or **global pruning** using a **error on holdout set** ,**statistical significance** or **MDL principle**.
For statistical validity evaluation must be done on data not used for **growing** the tree : requires **growing set** and **pruning set** (split from training set).
With the **reduced error pruning** the algorithm build the full tree on the growing set then prunes it. **Incremental reduced error pruning** simplifies **each** rule as soon as it is generated.

9.2 Indirect methods

Rules sets can be far more readable than decision trees (if DT are very complex) . Decision trees also suffer from **replicated subtrees** .
Rule sets are collection of **local models** while trees represents model over the **whole domain**. The covering algorithm focuses on **one class at a time** while DT focus on all classes together.

10 Other Classifiers

10.1 Naives Bayes Classifier

- **Conditional probability** :

$$P(C|A) = \frac{P(A \wedge C)}{P(A)}$$

$$P(A|C) = \frac{P(A \wedge C)}{P(C)}$$

- Bayes Theorem :

$$P(C|A) = \frac{P(A|C)P(C)}{P(A)}$$

A **prior probability** of C , $P(C)$, is the probability of an event before evidence is seen. A **posterior probability** of C , $P(C|A)$, is the probability of an event after evidence is seen. The probability of a **class** given an example:

- A s ample is a tuple of attributes

$$\overrightarrow{x} = \langle x_1, ..., x_n \rangle$$

- Given target y for the instance , find class with the **highest probability** for x:

$$class = argmax_y P(y|\overrightarrow{x})$$

Given the target y and sample x described by n attributes (assumed **statistically independent) for simplification** , the Bayes Theorem states that:

$$P(y|\overrightarrow{x}) = \frac{P(\overrightarrow{x}|y)P(y)}{P(\overrightarrow{x})} = \frac{P(\overrightarrow{x_1}|y)...P(\overrightarrow{x_n}|y)P(y)}{P(\overrightarrow{x})} =$$

- **Training**
  Computes the class probability $P(H)$ and the conditional probability $P(e_i|H)$ for each attribute value e_i and each class H.

- **Testing**
  Given an example E , computes the class:

$$class = argmax_y P(y|\overrightarrow{x}) = \frac{P(\overrightarrow{x_1}|y)...P(\overrightarrow{x_n}|y)P(y)}{P(\overrightarrow{x})}$$

$$= P(\overrightarrow{x_1}|y)...P(\overrightarrow{x_n}|y)P(y)$$

The classifier makes the assumptions that attributes are **independent** and **equally important**. Independent means that knowing the value of one attribute says nothing about the value of another : this is **often not true**! Example

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

$$
\begin{aligned}
P(yes|E) &= P(Sunny|yes)P(Cool|yes)P(High|yes)P(True|yes)P(yes) \\
&= \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14} \\
&= 0.0053
\end{aligned}
$$

$$
\begin{aligned}
P(no|E) &= P(Sunny|no)P(Cool|no)P(High|no)P(True|no)P(no) \\
&= \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \times \frac{5}{14} \\
&= 0.0206
\end{aligned}
$$

Conversion into a probability by normalization:

$$
\begin{aligned}
P(yes) &= 0.0053/(0.0053 + 0.0206) = 0.205 \\
P(no) &= 0.0206/(0.0053 + 0.0206) = 0.795
\end{aligned}
$$

The **Zero-Frequency Problem** occurs when an attribute value does not occur with every class , for example

$$\text{P(humidity} = \text{high } |yes) = 0$$

So no matter what the whole probability goes to zero.
A solution is to **add a constant term** to avoid this situation:

- add **1** to every count (**Laplace estimator**)
- add a constant $\neq 1$

$$\frac{2+\mu/3}{9+\mu} \quad \text{Sunny} \qquad \frac{4+\mu/3}{9+\mu} \quad \text{Overcast} \qquad \frac{3+\mu/3}{9+\mu} \quad \text{Rainy}$$

- add **weight** (can be different but must sum up to 1)

$$\frac{2+\mu p_1}{9+\mu} \qquad \frac{4+\mu p_2}{9+\mu} \qquad \frac{3+\mu p_3}{9+\mu}$$

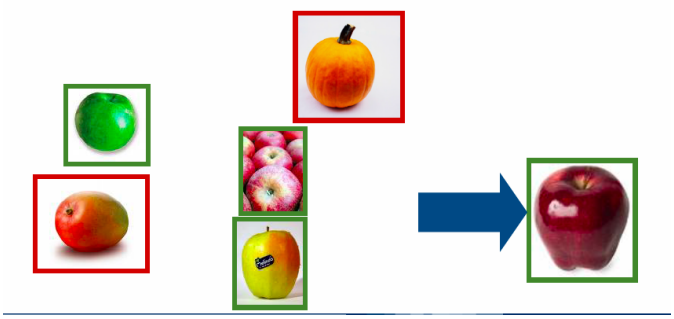
A **missing value** will not be counted during training , and will me **omitted** during testing .
Numeric attributes are assumed to have **Gaussian** or **Normal** distribution (given the class). The probability distribution function is defined by **mean** and **standard deviation**:

- **Sample mean** : $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$
- **Std Deviation** : $\sigma = \frac{1}{n-1}\sum_{i=1}^{N}(x_i - \mu)^2$
- **Density function** : $f(x) = \frac{1}{\sqrt{2\pi\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}}$

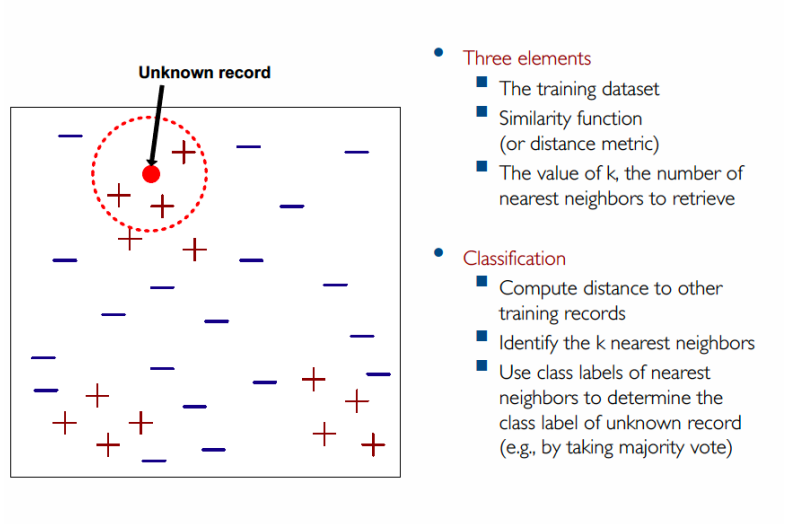
However often numeric attributes are **not normally distributed**. Also it is not always true that attributes are statistically independent : but this does **not** affect the performance of such a classifier as its performances are surprisingly good : classification does not require too much precision on probability estimates as long as the **maximum probability** is assigned correctly to a class.

10.2 Nearest Neighbour

Is a kind of **instance based learning**. A new sample is classified by looking for the k-most similar ones to the given sample and assigning it the majority class:



Instance based learning does **not compute** a model, but uses the **training records** to predict an unknown class label. This methods uses a **similarity measure** to define what is learned. K-Nearest-Neighbours implements this kind of instance based learning:



Choosing **k** correctly is important:

- **too small** : classification might be to sensitive to noise points
- **too large** : classification might include too dissimilar points

Also the choice of the **similarity measure** is important :

- **Euclidean distance**

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

  To determine the class from nearest neighbour just take the **majority vote** class label among the k neighbours or weight the vote according to distance.Other popular Euclidean Distances are:

  - **Lr Norm** $(\sum_i^n |x_i - y_i|^r)^{\frac{1}{r}}$
  - **Manhattan** $(\sum_i^n |x_i - y_i|$
  - **L infinite norm** $max_{i=1}^n |x_i - y_i|$

- **Cosine distance**
  The cosine distance between x,y is the **angle** that the vectors to the point

make

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

This angle will be between 0 and 180 no matter how many dimension there are.

- **Jaccard distance**
  Is a measure of how **dissimilar** two sets are :

$$d(\overrightarrow{x}, \overrightarrow{y}) = 1 - J(\overrightarrow{x}, \overrightarrow{y})$$

$$J(\overrightarrow{x}, \overrightarrow{y}) = \frac{|\overrightarrow{x} \cap \overrightarrow{y}|}{|\overrightarrow{x} \cup \overrightarrow{y}|}$$

- **Hamming distance**
  Hamming distance between two vectors is the number of components in which they differ or equivalently :

$$d(\overrightarrow{x}, \overrightarrow{y}) = \frac{n - m}{m}$$

  where n are the number of variables and m the number of matching components.

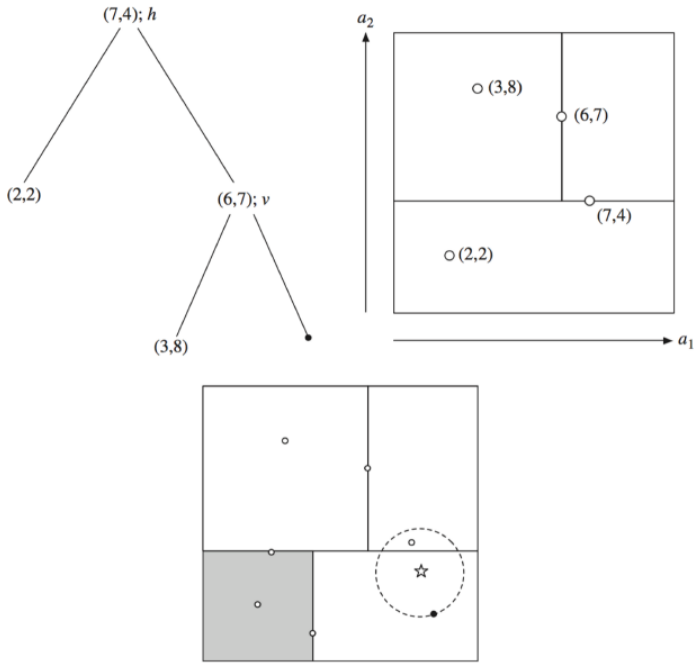
Another important step in KNN is **normalization** . Attributes are measured on **different** scales and need to be normalized using :

- **Instance range normalisation** : $x_i = \frac{x_i - min_i x_i}{max_i x_i - min_i x_i}$
- **standard score normalisation** : $x_i = \frac{x_i - \mu}{\sigma}$

For nominal attributes the value is 0 if they are not the same value , 1 if they are. Missing values are assumed to **maximally distant**

10.2.1 Improving KNN

KNN can be slow as it requires **linear scans** of the data.Classification time is proportional to the product of the number of instances.To speed this up there are different techniques like **KD-Trees**: they split the space hierarchically using a tree generated from the data. To find a neighbour simply navigate the tree



The tree is navigated KD Trees **don't care about the class** , they just partition the space.
Complexity depends on the **depth of the tree**,given by the logarithm of number of nodes for a balanced tree.
A good KD Tree :

- need to find good split and split direction
- possible split direction where the greatest variance is
- possible split point on the median value along that direction (mean can be used if data is **skewed**)

10.2.2 KNN Regression

KNN can also be used for **regression** , done in a **non-parametric** way (no learning of a model and weights) : KNN fits each point locally and computes the y_p value of a new point x_p by local **interpolating** the targets associated to neighbour points (prediction can use plain average or weighted average).

10.2.3 KNN Discussion

KNN is often very accurate but also very slow (speed up with KD Trees or Ball Trees). It assumes all attributes are equally important so considering an attribute selection first may improve the performances.
In case of **noisy** instance KNN should be applied using a majority vote over the k-nearest or remove noisy data (difficult!).

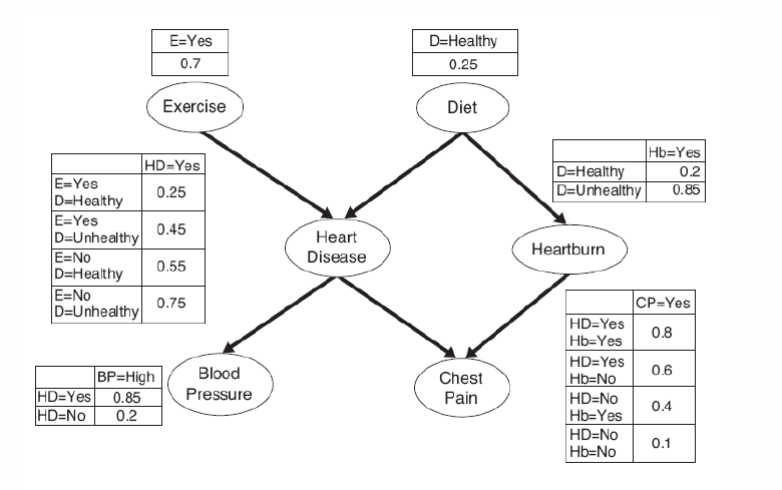
10.3 Bayesian Belief Networks

This model doesn't make all the assumptions of Naive Bayes Classifiers:

- BNN allow to **specify** which pair of attributes are **conditionally independent**.
- BNN provide a method to graphically represent **probabilistic relationships** among a set of variables

BNN describe the probability distribution governing a set of variables by using **conditional independence** assumptions on a subsets of variables and a set of **conditional probabilities**. So what is needed :

- **Acylic Graph** = encodes the relationships among variables
- **Probability table** = associating each node to its immediate parent node

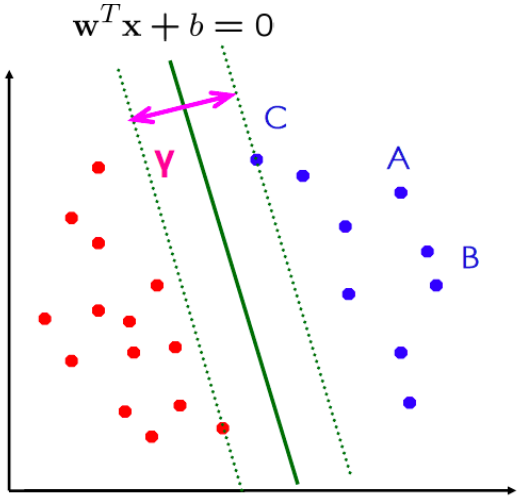


- If a node does not have a parent is contains the **prior probability** P(X) (like Exercise and Diet)
- If a node has just one parent it contains the conditional probability $P(X|Y)$
- If a node has more parents it contains the conditional probability $PX|Y_1, ..., Y_k$

BNN are very useful to incorporate **domain knowledge** into the model (other model cannot do this!). BNN's are very time consuming but adding variables is very easy. They are robust to overfitting and also work on incomplete data.

10.4 Support Vector Machines

Two classes can be separated in many ways : SVMs work by **maximizing** the margin :

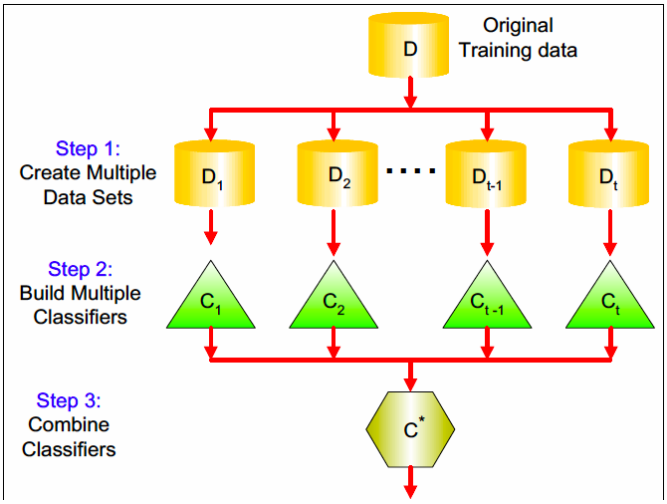


SVMs work by searching for the hyperplane that maximizes the margin or the largest γ such that

$$\forall i, y_i(wx_i + b) \geq \gamma$$

11 Classification Ensembles

Ensembles generate a **set of classifiers** from the training data.They predict the class label by **aggregating** predictions made from **previous classifiers**



Using more classifiers and let them vote to decide the outcome often increases performance greatly at the cost of creating hard to analyze structures and outputs. Assume there are 25 independent classifiers each with error rate $\epsilon = 0.35$ The probability that the ensemble makes an error is :

$$\sum_{i=1}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$

How can classifiers be independent is they train on the same dataset?

11.1 Bagging

Given a set D of d tuples, at each iteration i ,a training set D_i of tuples is sampled with replacement from D (**bootstrap**).Then a **classifier** M_i is learned for each training set D_i.
Prediction is performed by taking all outputs of the classifier and assigns the class with the **most votes**. This gives equal weight to each classifier.
This approach can also be applied to **regression** by taking the **average of the votes**.
Bagging works because it **reduces the variance** by voting/averaging but **cannot always be applied**:

- if the learning algorithm is **unstable** (small changes in training set cause big changes in classifier) ,bagging almost always increases performance. Good for NN ,DT, Regression Trees, Linear Regression...

- if the learning algorithm is stable (example : KNN) using bagging is a bad idea.

11.2 Algorithm Randomization

Instead of randomizing data , the algorithm itself can sometimes be randomized:

- weights in NN can be randomized
- attribute selection or confidence level for pruning in DT can be randomized. The more **uncorrelated the trees** the greater the **variance reduction**.
- ...

Often randomizing the algorithm can be used in combination with **bagging**.

11.2.1 Random Forests

It is a learning ensemble consisting of a **bagging** of unpruned **decision tree learners** with **randomized selection of features** at each split. The **generalization error** of a RF depends on the **individual strength** of the trees and the **correlation** between them.

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Random forest have an amazing feature called **Out-of-bag Evaluation** , a method to evaluate performance without using **cross-validation** : for each observation (x_i, y_i) construct its random forest predictor by **averaging** only those trees corresponding to bootstrap samples in which the observation **did not appear**. This can be done during the **fitting phase**!
RF are very easy too use, as they require only few parameters to set up , but have a **high accuracy** and are good ad avoiding **overfitting** by choosing a high number of trees. Additional properties:

- Returns variable importance estimates
- A excellent in classification , not so excellent in regression
- Gains accuracy if used with boosting

11.3 Boosting

Boosting consists in making a final prediction working on the errors of previous learners. The first boosting method used is **AdaBoosting**.

1. Assign weights to each training example
2. Learn iteratively k classifiers
3. After classifier M_i is learned the weights are updated to allow the subsequent classifier $M_{i+1}t$ to pay more attention to the training tuples that were missclassified by M_i.
4. The final M_* combines the votes of each individual classifier where the weight of each classifier's vote is a function of its accuracy.

Boosting work with **weak classifiers** , as long as the error **does not exceed 0.5**. Also boosting **can lead to overfitting**.

11.3.1 AdaBoosting

AdaBoost computes a **strong classifier** as a combination of a week classifier

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

where

- $h_t(x)$ is the output of weak classifier t
- α_t is the weight assigned to classifier t from its **estimated error** ϵ_t:

$$\alpha_t = \frac{1}{2} ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

The weights are update by multiplying them by :

- $e^{-\alpha}$ if correctly classified
- e^{α} if incorrectly classified

While the error rate is computed as :

$$\epsilon_i = \frac{1}{N} \sum_{j=1}^{N} w_j \delta(h_i(x_j) \neq y_i)$$

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Example of adaboost

- At the beginning, all the data points have the same weight and thus the same probability of being selected for the training set. With 10 data points the weight is 1/10 or 0.1

w	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1

- Suppose that in the first round, we generated the training data using the weights and generated a weak classifier h_1 that returns +1 if x<3

h_1

w	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1
	+1	+1	+1	-1	-1	-1	-1	-1	-1	-1

- The error ϵ_1 on the weighted dataset is 2/10 and α_1 is 0.69
- The weights are updated by multiplying them with
  - $\exp(-\alpha_1)$ if the example was correctly classified
  - $\exp(\alpha_1)$ if the example was incorrectly classified
- Finally, the weights are normalized since they have to sum up to 1

- The new weights are thus,

w	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.25	0.25
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1

- Suppose now that we computed the second weak classifier h_2 as

h_2

w	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.25	0.25
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1
	-1	-1	-1	-1	-1	-1	-1	-1	+1	+1

- That has an error ϵ_2 of 0.1875 and α_2 is 0.7332

And so on.

11.3.2 Boosting Stumps

Boosting can also be used in a Random Forest contest. A weak learner in this contest is the **decision stump** ,a simple one node DT. A good thing in boosting is that it continues to reduce the **test error** even if the training error is **already 0** :this suggests that boosting does not use the training error as basis for learning its trees.

11.3.3 Gradient Boosting

Easier to understand for regression but largely used in classification. In summary:

1. Learn a regression predictor
2. Compute the error residual
3. **Learn to predict the residual**
4. Repeat 2-4

The sum of predictions is increasingly accurate while the predictive function get more and more complex. Example on the MSE:

- To compute a model to predict target value y_i that minimizes a loss function like MSE

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Adjust $\hat{y}_i$ to try to reduce the error using gradient

$$\hat{y}_i = \hat{y}_i + \alpha \nabla MSE(y, \hat{y})$$

  Where the gradient is a function of $y_i - \hat{y}_i$

Each learner tries to estimate the gradient for the loss.

```
# start with the basic predictor (the mean)
mu = mean(y)
dY = y - mu

# number of models
n_boost = 50

for k in range(n_boost):

    # fit the residual
    learner[k] = model.fit(X,dy)

    # set the learning rate
    alpha[k] = 1

    # update the residual
    dy = dy - alpha[k] * learner[k].predict(X)
```

```
# X_test and y_test are the test data
predict = zeros(len(y_test))

for k in range(n_boost):
    predict = predict + alpha[k]*learner[k].predict(X_test)
```

11.3.4 eXtreme Gradient Boosting

Efficient and scalable implementation of gradient boosting applied to classification and regression trees. **Only deals with numerical variables** (use **CatBoosting** for categorical data).
Its main advantages are that it is very fast wrt to normal gradient boosting as it uses:

- novel tree learning algorithm to handle **sparse data**
- approximate learning using **quantile sketch**
- more regularized model formalization to control **overfitting**

11.4 Learning Ensembles

Learning ensembles steal the idea behind RF and Boosting by building **stonger models** by performing a global optimization over the ensemble.

- Generate a **dictionary of models** :

$$D = \{T_1(X), T_2(X), ..., T_M(X)\}$$

- Fit a model to the data

$$F(X) = \sum_{\alpha \in D} \alpha_m T_m(X)$$

Stacking generalization train a learning algorithm to combine predictions of **heterogeneous** set of learning algorithms:

- Train a set of base learners over the data
- Use a **meta learner** that is trained using the predictions of the base classifiers as additional inputs

Apply CV

12 Clustering

12.1 Intro

Clustering is an **unsupervised** (no predefined class) machine learning technique that **groups** a collection of data point into **clusters** according to some **distance measure** so that data points in the cluster have **a small distance** from each other and data points in different clusters should be at large distances.
Good clustering consists of :

- High intra-class similarity
- Low inter-class similarity

The quality of clustering depends on the **similarity measure** (but also by its **implementation**) and its ability to find some or all **hidden patterns**. To evaluate the goodness of clustering :

- Various measures of similarity
- Manual inspection
- Benchmark with existing labels

The similarity is usually expressed as a function of **distance**,typically metric

$$d(i, j)$$

. This distance functions depend on the variables used , as they differ for interval-scaled,boolean, categorical ,ordinal ratio, and vector variables. The **goodness** of clustering however must be measured by **another function**. The concept of **good enough** is very **subjective**!

12.1.1 Clustering methods

- Hierarchical vs point assignment
- Numeric and/or symbolic data
- Deterministic vs non deterministic

- Exclusive vs overlapping
- Hierarchical vs flat
- Top-down vs bottom-up

12.1.2 Data structures

- **Data matrix**

$$\begin{bmatrix} x_{11} & ... & x_{1f} & ... & x_{1p} \\ ... & ... & ... & ... & ... \\ x_{i1} & ... & x_{if} & ... & x_{ip} \\ ... & ... & ... & ... & ... \\ x_{n1} & ... & x_{nf} & ... & x_{np} \end{bmatrix}$$

- **Dis/Similarity Matrix**

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ ... & ... & ... & ... & ... \\ d(n,1) & d(n,2) & ... & ... & 0 \end{bmatrix}$$

12.1.3 Distance and Similarity Measures

The measures are very similar to the ones used in KNN:

- L_r norm
- Euclidean distance ($r = 2$)
- Manhattan distance ($r = 1$)
- L_∞ norm
- Jaccard distance
- Cosine distance

- Hamming distance
- Edit distance (new)
  The distance between a string x=$x_1x_2...x_n$ and y =$y_1y_2...y_m$ is the **smallest number of insertions and deletion** of single character that **transform x into y** or can be defined as the **longest common subsequence LCS** of x and y and then
  $$d(x, y) = |x| + |y| - 2|LCS|$$

12.1.4 Requisites for clustering

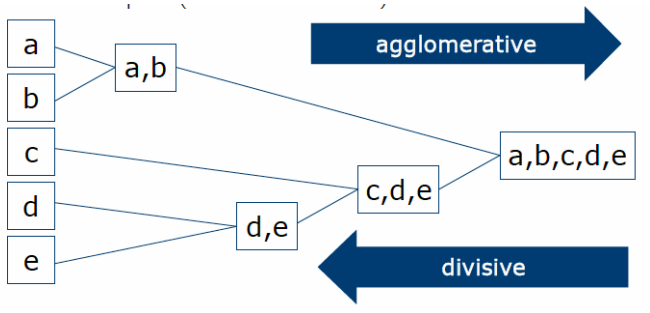
- Scalability
- Ability to deal with different types of attributes
- Ability to handle data dynamically
- Discovery of clusters dynamically
- Discovery of clusters with arbitrary shape
- Able to deal with **noise** and **outliers**
- Incorporate user specified constraints
- High dimensionality
  Curse of dimensionality: in high dimensions ,almost all pair of points are **equally far away from one another** $\rightarrow$ **orthogonal vectors**
- Insensitive to order of inputs
- Interpretability and usability

12.2 Hierarchical clustering

One of the most common types of clustering. Can be done in two ways:

- **Agglomerative**
  Start at **individual clusters**, at each step ,merge the **closest pair** of clusters until **one cluster** o **k-clusters** left

- **Divisive**
  Start with **one cluster** ,at each step , **split a cluster** until each cluster contains a point or there are k-clusters.



Easy to used and any desired number of cluster can be obtained by **cutting** the dendrogram at a proper level. Traditional hierarchical clustering algorithms use **similarity** or **distance matrix** to merge or split **one cluster at the time**.
The most popular is the **agglomerative** algorithm :

1. Compute proximity matrix
2. Let each data point be a cluster
3. Repeat until single cluster remains :
   - Merge two **closest** clusters
   - **Update** proximity matrix

Key operation is to define the **proximity** matrix between two clusters : different algorithm use different measures and lead to different clusters.

12.2.1 Complexity

- $O(N^2)$ space since it uses the **proximity matrix**
- $O(N^3)$ in many cases : there are N steps at each step of size N^2 (proximity matrix must be searched and updated)

Efficient implementation :

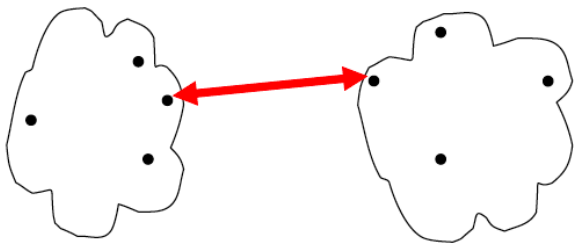
- Compute distance between all points $O(N^2)$
- Insert pairs and their distances in a **priority queue** to find the min one in one step $O(N^2)$
- When merging two clusters, **remove all entries** in the queue involving one of these 2 clusters $O(NlogN)$
- Compute the distance between the cluster and the remaining clusters $O(NlogN)$
- Since the complexity of the **last two steps** is repeated **at most N times** ,total complexity is

$$O(N^2logN)$$

12.2.2 Distance between clusters

When merging two cluster who is the proximity matrix updated?

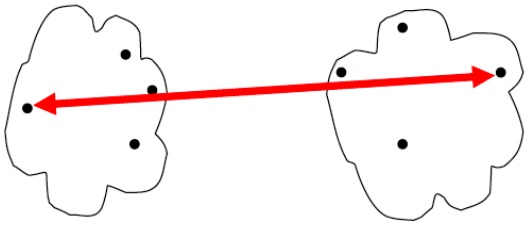
- **Single linkage or MIN**



Takes the **smallest distance** between an element of one cluster and element in the other

$$d(C_i, C_j) = min(t_{i,p}, t_{j,p})$$

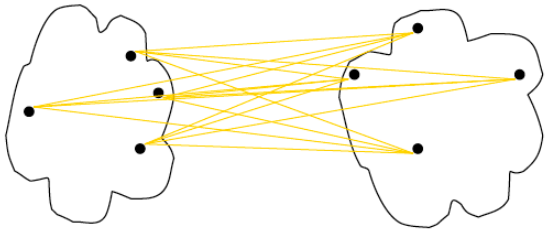
- **Complex linkage or MAX**



Take the **largest distance** between an element in one cluster and an element in the other

$$d(C_i, C_j) = max(t_{i,p}, t_{j,p})$$

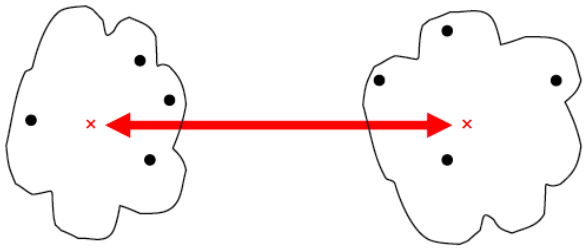
- **Average or Group Average**



Take the **average distance** between an element in one cluster and an element in the other

$$d(C_i, C_j) = avg(d(t_{i,p}, t_{j,p}))$$

- **Centroid**



Take the **largest distance** between an element in one cluster and an element in the other

$$d(C_i, C_j) = d(\mu_i, \mu_j)$$

where the μs are **centroids**

12.2.3 Determining number of clusters

HC generates a set of N possible partitions ,which one should be used? A **good cluster** as already said should have small intra class distance and high inter class distance.
A good measure is the **Within/Between Cluster Sum of Squares**:

- **Within cluster SoS**:

$$WSS(C) = \sum_{I=1}^{k} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$

where μ_i is the **centroid** of cluster C_i.
For a distance function **d**:

$$WSS(C) = \sum_{I=1}^{k} \sum_{x_j \in C_i} d(x_j - \mu_i)^2$$

- **Between-cluster SoS**:

$$BSS(C) = \sum_{i=1}^{K} |C_i|||\mu - \mu_i||^2$$

where μ is the centroid of the **whole dataset**.
For a distance function **d**:

$$BSS(C) = \sum_{i=1}^{K} |C_i| d(\mu - \mu_i)^2$$

Then a **knee/elbow analysis** is performed on the plots to show **significant** modifications in the evaluation metrics.

12.2.4 Cluster representation

In **Euclidean space** a cluster can be identified using for example its **centroid** or use its **convex hull**. In **Non-Euclidean** spaces a distance can be defined (jaccard,cosine,edit) and the concept of **clusteroid** can be used : an **existing data point** used as **representative** for the cluster (can be the point that minimizes the sum of squares of the distances). Another way to represent clusters if to check the **feature distribution** in the cluster and check if they are different.

12.2.5 Problems with HC

- Once a cluster merge has been done it cannot be undone
- Sensitive to outliers and noise
- No direct objective function to be minimized
- Difficulty in breaking large clusters
- Agglomerative method is commonly used by does not scale very well (at least $O(N^2)$ complexity)

12.3 Representative based Clustering

The algorithms work around building the **representative** for each cluster . Given a set of N instances and a desired number of k, this algorithms generate a partition C of N in k clusters :

$$\{C_i, ..., C_k\}$$

And for each cluster there is a **point** that **summarizes** the cluster. A common choice is to use the **mean**

$$\mu_i = \frac{1}{n_i} \sum_{x_i \in C_i} x_i$$

with μ_i being the **centroid**. The goal is to select the **best partition** according to some scoring function, for example the **sum of squared errors** :

$$SSE(C) = \sum_{I=1}^{k} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$

$$C^* = argmin_C SSE(C)$$

Brute force approach :

$$O(\frac{k^N}{k!} \text{ possible paritions}$$

12.3.1 K-Means Clustering

Most used representative based algorithm. It assumes an **euclidean** space can be easily extended to a non-euclidean case. The algorithm employs a **greedy** iterative approach that minimizes the SSE which can lead to find optimal solution but also often to sub-optimal solutions.

1. Choose k points that are likely to be in different clusters
2. Make these points the centroids of their clusters
3. For each remaining point :
    - find centroid to which the point is closest
    - add the point to the cluster
    - adjust centroid

K-MEANS $(\mathbf{D}, k, \epsilon)$:

$t = 0$

Randomly initialize k centroids: $\boldsymbol{\mu}_1^t, \boldsymbol{\mu}_2^t, \ldots, \boldsymbol{\mu}_k^t \in \mathbb{R}^d$

repeat

  $t \leftarrow t + 1$

  $C_j \leftarrow \emptyset$ for all $j = 1, \cdots, k$

  // Cluster Assignment Step

  foreach $\mathbf{x}_j \in \mathbf{D}$ **do**

    $j^* \leftarrow \arg\min_i \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_i^t \right\|^2 \right\}$ // Assign $\mathbf{x}_j$ to closest centroid

    $C_{j^*} \leftarrow C_{j^*} \cup \{\mathbf{x}_j\}$

  // Centroid Update Step

  foreach $i = 1$ *to* k **do**

    $\boldsymbol{\mu}_i^t \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$

until $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$

Cluster initialization

The initialization of the clustering is important :

- **Solution 1**
  Pick points that are as far away as possible from each other or pick the **first** point at random and while there are fewer than K points add the points whose **minimum distance** from the selected points is as **large as possible**.
- **Solution 2**
  Cluster a sample of data (hierarchically ?) so there are k-clusters. Pick a point from each cluster (the one closest to the centroid?)

If there are K real clusters then the chance of selecting on centroid from each cluster is **small** (especially when K large).If clusters are the same size n:

$$P\frac{\text{number of ways to select one centroid from each cluster}}{\text{number of ways to select K centroid}} = \frac{k!n^k}{(kn)^k} = \frac{K!}{K^k}$$

So picking the right centroids is very hard and not always will they **converge** to the ideal centroid!
To overcome this issue:

- Multiple re-runs
- Sample and use another sampling method
- Select more than k-initial centroids and then select among these

- Post-processing
- **Bisecting k-means**
  Can produce a partitional or hierarchical clustering

Algorithm 3 Bisecting K-means Algorithm.

```
1: Initialize the list of clusters to contain the cluster containing all points.
2: repeat
3:    Select a cluster from the list of clusters
4:    for i = 1 to number_of_iterations do
5:       Bisect the selected cluster using basic K-means
6:    end for
7:    Add the two clusters from the bisection with the lowest SSE to the list of clusters.
8: until Until the list of clusters contains K clusters
```

Cluster number

Also the choice of the **number of clusters** must be made. A solution is to use the **Knee/Elbow Analysis**.

Centroid update

Centroid can be updated:

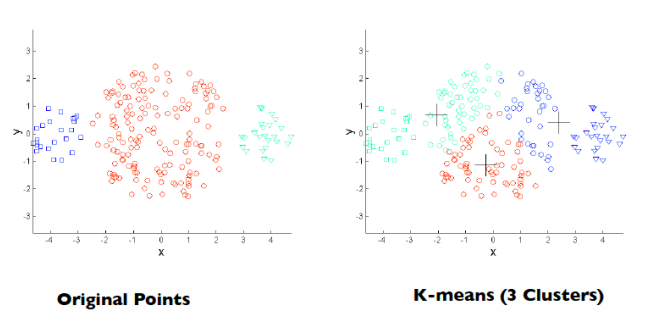
- When all points are assigned (usual method)
- **Incrementally** , very expensive as each points updates zero or two centroids!

Pre and Post Processing

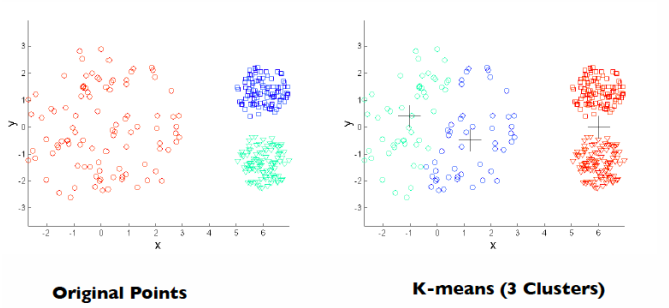
- **Pre-processing** : **normalize data** + **eliminate outliers**
- **Post processing**: eliminate **small clusters**, **split** loose clusters (relatively high SSE) , **merge** close clusters (relatively low SSE)

Limitations of clustering

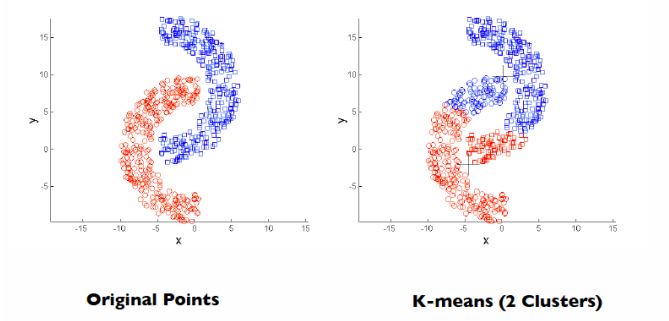
- **Differing Sizes**



- **Different densities**



- **Non globular shapes**



A solution to these can be to use **many clusters** and merge them (bigger k than expected).

Summary

- **Positive** :efficient , terminates often in optimal solution using annealing or genetic algorithms

- **Negative** : only works when mean can be defined (no cat) , need to specify k in advance , bad with noisy data, not usable with cluster in non-convex shape

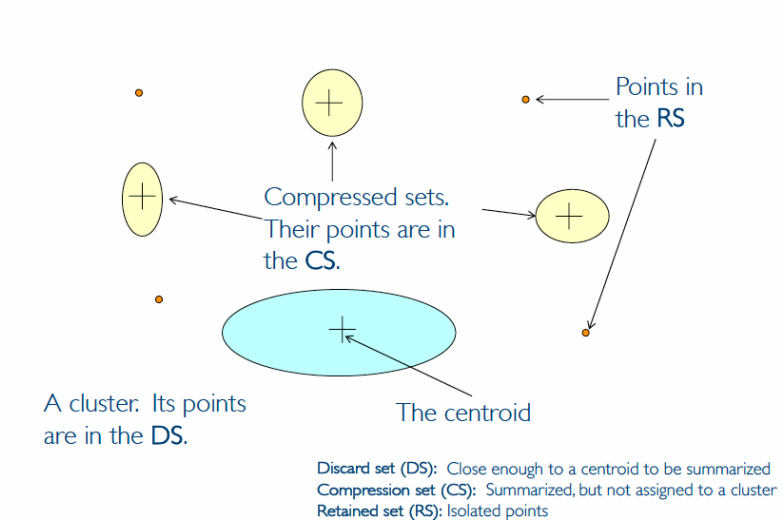
12.3.2 BFR

Extension to K-Means, can handle much more data (disk-resident data). It assumes that **data is normally distributed** around a **centroid**.Clusters are **axis-aligned ellipses**.

1. Points are read from disk one chunk at the time
2. Most points from previous memory loads are summarized by simple statistics
3. Select k initial centroids by sensible approach (k random points, take small sample and cluster optimally , take sample + random point then k-1 more points each as far away as possible)

Points are divided into three sets:

- **Discard set** = points close enough to a centroid to be summarized
- **Compression set** = groups of points that are close together but not close to any existing point. These are **summarized** but **not assigned to a cluster**
- **Retain set** = Isolated points waiting to be assigned to a compression set



The **discard set** is summarized by:

- **N** the number of points
- **Vector SUM** whose component sum(i) is the sum of the coordinates of the points in the i-th dimension.
- **Vector SUMQ** whose component sumq(i) is the sum of the square coordinates of the points in the i-th dimension.
- 2 —dimension— + 1 represent the size of any cluster
- Average in each dimension $\frac{SUM(i)}{N}$
- Variance of cluster's discard set in dimension i $(\frac{SUMSQ(i)}{N}) - (\frac{SUM(i)}{N})^2$

How the algorithm works :

1. First all points **sufficiently close** to the centroid of a cluster are added to that cluster (update points parameters) then discard the points
2. Points not close enough are clustered with the points in the **retain set** (can use any clustering algorithm)
3. Minicluster derived from new points + old retain set are merged
4. Any point outside a cluster is dropped

A point is **sufficiently close** then it is added using two approaches :

1. Add p to cluster if it has the centroid closest to p and it is unlikely that after all points have been processed another centroid will be found closer to p
2. Measure probability that if p belongs to a cluster , it would be found as far as it is from the centroid of the cluster

In practice the **Mahalanobis distance** is used : it computes the distance between a point and the centroid of a cluster,normalized by the standard deviation of the cluster in each dimension

$$\sqrt{\sum_{i=1}^{d} \left(\frac{p_i - c_i}{\sigma_i} \right)^2}$$

A point is assigned to a cluster if **below a certain threshold**. For example threshold 4 means that there is a chance in a million not to include something that belongs to the cluster!

12.3.3 Expectation Maximization

K-means assigns each point to only one cluster **(hard assignment)**. Extend approach to consider **soft assignment** of points to clusters. Assume that each cluster C_i is characterized by a multivariate **normal distribution** :

- mean vector μ_i
- covariance matrix Σ_i

Clustering is defined by a vector of parameters Θ :

$$\Theta = \{\mu_i \Sigma_i P(C_i)\}$$

The goal of MLE is to choose parameters so that :

$$\Theta^* = argmax_\theta P(D|\Theta)$$

1. Start with initial estimate of vector parameter
2. Rescore the pattern against a mixture density produced by parameter vector
3. Rescored patterns used to update parameter vector
4. Patterns belonging to the same cluster, if they are placed by their scores in a particular component

Maximize the probability that the data comes out of the found distribution :

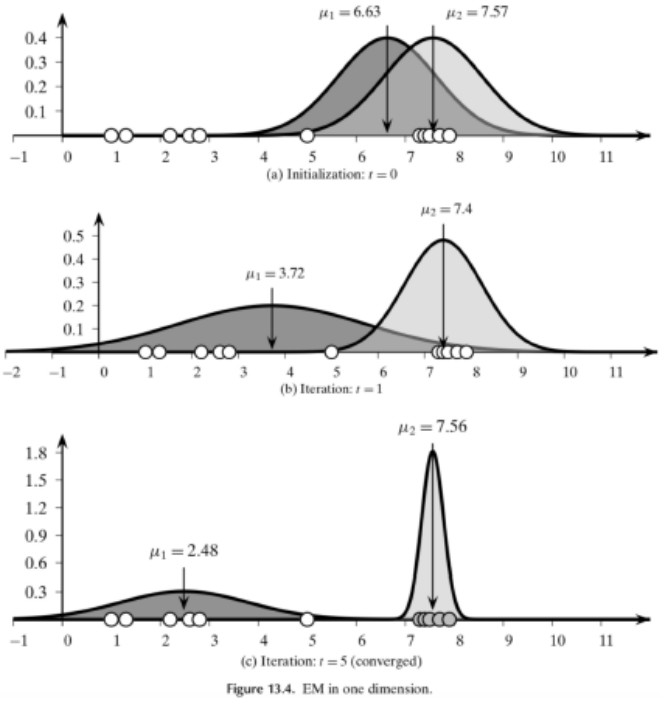


Figure 13.4. EM in one dimension.

12.4 Density Based Clustering

Density based clustering is a clustering technique based on density ,local cluster criterion , such as density connected points. It is useful as it can :

- Discover cluster of arbitrary shape (flexible)
- Handle noisy data (robust)
- Perform one scan (efficient)
- Need density parameters as termination condition

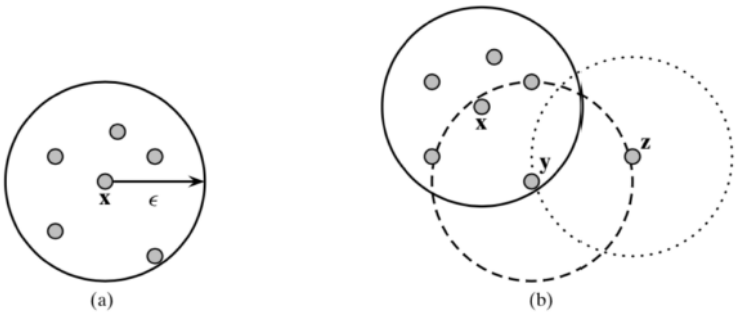
Many interesting studies like **DBSCAN**:

- ϵ - neighbourhood of an object :

$$N_\epsilon(x) = \{y | \delta(x, y) \leq \epsilon\}$$

  Is the neighbourhood with radius ϵ of a given object

- **Core object** : if the ϵ - neighbourhood of an objects contains at least **minpts** then the object is a **core object**
- **Directly density reachable** : an object x is directly density reachable from object y if x is within the ϵ-neighbourhood of y and y is a core object
- **Density reachable** : an object x is density reachable from y if there is a chain of object $x_1, ..., x_n$ where $x_1 = x$ and $x_n = y$ such that x_{i+1} is directly reachable from x_i
- **Density connected**: an object p is density-connected to q with respect to ϵ and MinPts if there is an object o such that both p and q are density reachable from o
- **Density Based Cluster**: a density based cluster is defined as a maximal set of density connected points



- **Density** = have at least minpts within ϵ
- **Border** = fewer than minpts within ϵ but in neighbourhood of a **core point**
- **Noisy** = is not a core point neither a border point

ALGORITHM 15.1. Density-based Clustering Algorithm

```
DBSCAN (D, ϵ, minpts):
Core ← ∅
foreach x_i ∈ D do // Find the core points
    Compute N_ϵ(x_i)
    id(x_i) ← ∅ // cluster id for x_i
    if N_ϵ(x_i) ≥ minpts then Core ← Core ∪ {x_i}
k ← 0 // cluster id
foreach x_i ∈ Core, such that id(x_i) = ∅ do
    k ← k + 1
    id(x_i) ← k // assign x_i to cluster id k
    DENSITYCONNECTED (x_i, k)
C ← {C_i}_{i=1}^k, where C_i ← {x ∈ D | id(x) = i}
Noise ← {x ∈ D | id(x) = ∅}
Border ← D \ {Core ∪ Noise}
return C, Core, Border, Noise

DENSITYCONNECTED (x, k):
foreach y ∈ N_ϵ(x) do
    id(y) ← k // assign y to cluster id k
    if y ∈ Core then DENSITYCONNECTED (y, k)
```

DBSCAN fails if the data:

- has **varying densities**
- is **highly dimensional**

12.5 Cluster Validation

12.5.1 Cluster Evaluation

Cluster evaluation assesses the **goodness** or **quality** of the clustering.

1.External Validation Measures

Employ criteria that are **not inherent** to the dataset (**prior** or **expert knowledge** about the clusters , like class labels for each point). Here the **ground-truth** is known a-priori.

1. **Using TP,FP,TN,FN**

   - **TP** = number of true positive pairs. A pair x_i, x_j are TP if they belong to the same **true partition T** and they are also in the **same cluster C**.

- **FN** = number of false negative pairs. A pair x_i, x_j are FN if they belong to the same **true partition T** and they are also in the **same cluster C**.
- **FP** = number of false positive pairs. A pair x_i, x_j are FP if they do not belong to the same **true partition T** but they are in the **same cluster C**.
- **TN** = number of true negative pairs. A pair x_i, x_j are TN if they do not belong to the same **true partition T** nor are they in the **same cluster C**.
- **N = TP+TN+FP+FN**

These measures allow to specify different **coefficients**:

- **Jaccard Coefficient** :

$$\frac{TP}{TP + FP + FN}$$

, ignores the **true negatives**. Maximum value is **1** (no FP , no FN).

- **Rand Statistic** :

$$\frac{TP + TN}{N}$$

,measures all points were T and C agree.

- **Fowlkes-Mallows Measure** :

$$\sqrt{\text{precision} \cdot \text{recall}}$$

$$\text{prec} = \frac{TP}{TP + FP}$$

$$\text{rec} = \frac{TP}{TP + FN}$$

---

2. **Mutual Information Based Scores**

Tries to quantify the amount of **shared information** between the clustering C and T

$$I(C, T) = \sum_{i=1}^{r} \sum_{j=1}^{k} p_{i,j} log \left(\frac{p_{ij}}{p_{C_i} \cdot p_{T_j}} \right)$$

where p_{ij} is the probability that a point belong to cluster i and ground-truth partition j , p_{C_i}, p_{T_j} are the probabilities of cluster C_i and partition T_j.
The **Normalized Mutual Information** score is :

$$NMI(C,T) = \sqrt{\frac{I(C,T)}{H(C)} \cdot \frac{I(C,T)}{H(T)}} = \frac{I(C,T)}{\sqrt{H(C) \cdot H(T)}}$$

where $H_C = -\sum_{i=1}^{r} p_{C_i} log p_{C_i}$ (same formula for H(T).
Values close to **0** indicate label assignments that are **largely independent** , close to **1** indicate **significant agreement**.

---

3. **Homogeneity ,Completeness ,V-Measure**

   - **Homogeneity** : each cluster contains only members of a **single class**
   - **Completeness**: all members of a given classa are assigned **to the same cluster**
   - **V-Measure**: **harmonic mean** of homogeneity and completeness

   All between $[0, 1]$, the higher **the better**.

2.Internal Validation Measures These are criteria derived from the **data itself**.These measures are based on the notion of **intracluster similarity** contrasted with the notion of **intercluster separation** : they usually provide a **trade-off** to maximize these two contrasting measures.

- Sum over all the **intracluster** weights over all the cluster

$$W_{in} = \frac{1}{2} \sum_{i=1}^{k} W(C_i, C_i)$$

- Sum over all the **intercluster** weights

$$W_{out} = \sum_{i=1}^{k-1} \sum_{j>1} W(C_i, C_j)$$

- Number of distinct **intracluster edges** N_{in} and **intercluster edges** N_{out}:

$$N_{in} = \sum_{i=1}^{k} \binom{n_i}{2} = \frac{1}{2} \sum_{i=1}^{k} n_i(n_i - 1)$$

$$N_{out} = \sum_{i=1}^{k-1} \sum_{j>i}^{k} n_i \cdot n_j = \frac{1}{2} \sum_{i=1}^{k} \sum_{j \neq i, j=1}^{k} n_i \cdot n_j$$

- **BetaCV**

  Ratio of mean of intracluster distance to the mean of intercluster distance

$$\text{BetaCV} = \frac{W_{in}/N_{in}}{W_{out}/N_{out}} = \frac{N_{out} \sum_{i=1}^{k} W(C_i, C_i)}{N_{in} \sum_{i=1}^{k} W(C_i, \bar{C}_i)}$$

  The smaller this ratio **the better** : it indicates that on average the intracluster distances **are smaller** than intercluster distance

- **C-Index**

  Let $W_{min}(N_{in})$ be the sum of the **smallest** N_{in} distances in the proximity matrix W , where N_{in} is the total number of intracluster edges. Let $W_{max}(N_{in})$ be the sum of the **largest** distances in W. The **C-Index** measures to what extend the clustering puts together the N_{in} points that are closest to the k clusters

$$C - index = \frac{W_{in} - W_{min}(N_{in})}{W_{max}(N_{in}) - W_{min}(N_{in})}$$

  The **smaller** the **better** : indicates more compact clusters with relatively smaller distances within clusters rather than **between clusters**.

- **Dunn-Index**

  Ratio between minimum distance between points pairs from different

clusters and the maximum distance between point pairs from same cluster

$$\text{Dunn} = \frac{W_{out}^{min}}{W_{in}^{max}}$$

The **larger** the **better**: indicates that even the closest distance between points of different clusters is **much larger** than the farthest distance within same cluster.

- **Davies-Bouldin Index**

  The cluster mean is μ_i .The dispersion around the mean

$$\sigma_\mu = \sqrt{\frac{\sum\limits_{x_j \in C_i} \delta(x_j, \mu_i)^2}{n_i}} = \sqrt{var(C_i)}$$

  The measure , for a pair of cluster i and j, is:

$$DB_{ij} = \frac{\sigma_{\mu_i} + \sigma_{\mu_j}}{\delta(\mu_i, \mu_j)}$$

  This measure checks how **compact** the cluster is with respect to the **distance between the cluster means**.

  The index is finally defined as :

$$\text{DB-Index } = \frac{1}{k} \sum_{i=1}^{k} max_{j \neq i} \{DB_{ij}\}$$

  For each cluster i the cluster j is picked that **yields the largest DB ratio**.

  The **smaller** the better: indicates that clusters are **well separated** and each cluster is well represented by its mean.

- **Silhouette Coefficient**

  Measures both cohesion and separation of clusters ,based on the difference between the average distance points int the closest cluster and to points in the same cluster

$$s_i = \frac{\mu_{out}^{min}(x_i) - \mu_{in}(x_i)}{max\{\mu_{out}^{min}, \mu_{in}(x_i)\}}$$

  where

$$\mu_{in} = \frac{\sum\limits_{x_j \in C_{\hat{y}}, i \neq j} \delta(x_i, x_j)}{n_{\hat{y}_i} - 1}$$

$$\mu_{out}^{in}(x_i) = min_{j \neq \hat{y}_i} \left\{ \frac{\sum\limits_{y \in C_j} \delta(x_i, y)}{n_j} \right\}$$

The values of s_i lies between $[-1, +1]$:
- close to **1** : x_i is much closer to points in its own cluster and far from other
- close to **0** : close to boundary between two clusters
- close to **-1** : x_i is much closer to another cluster than to its own

The silouhette coefficient is defined as:

$$SC = \frac{1}{n} \sum_{i=1}^{k} s_i$$

The **closer** to **1** the **better**.
This measure should be used for convex clusters and not for DBSCAN for example

- **Calisinki-Harabaz Index**
  Given k-clusters CH-score s is given by the ratio of the **between cluster dispersion mean** and the **within cluster dispersion mean**:

$$C = \frac{BSS(C)/(k-1)}{WSS(C)/(N-k)} = \frac{(N-k)BSS(C)}{(k-1)WSS(C)}$$

  The **higher** the **better** : indicates that clusters are well separated.
  The score is higher for convex cluster and lower for density based clusters.

3.Relative Measures

Compare different clusterings obtained by varying different parameters for the same algorithm (e.g. number of clusters k).

- Knee/Elbow Analysis

- Calisinki-Harabaz Index
  Can be used to tare k :

$$\delta(k) = (CH(k+1) - CH(k)) - (CH(k) - CH(k-1))$$

- Silhouette Coefficient
  s_i can be used to of each point x_i and the average SC to estimate the number of clusters in the data : for each cluster plot s_i values in **descending order**

12.5.2 Cluster Stability

Clusterings obtained from **several datasets** sampled from the same distribution should be **similar** or **stable**

12.5.3 Cluster Tendency

Aims to determine whether the dataset has any **meaningful groups to begin with**.Difficult task, tackled by comparing the data distribution with samples randomly generated from the same data space. Existing approaches use **spatial histograms**, **distance distribution**, **Hopkins statistic**

13 Data Preparation

Data preprocessing is important because the quality of data mining depends on the quality of the data . For example duplicates or missing values can lead to **wrong statistics** about the data. Data warehouses need **consistent integration** of quality data (it's actually the most time-consuming task in data warehouses).

13.1 Data Cleaning

- **Missing Values**
  Values can missing because the information was **not collected** (e.g.: people not declaring age) or because attributes are **not applicable** to all cases (e.g.: annual income not applicable to children).
  These values can be **eliminated**, **imputated** , **ignored**...

- **Duplicate data**
  Dataset may include duplicate data or almost duplicates of one another.This can be an **issue** when **merging data** from heterogeneous source (e.g. : same person with multiple email addresses)

- **Outliers**
  Outliers are data objects with characteristics that are considerably different than most of the other data objects in the dataset.

13.2 Sampling

Sampling is the main data technique used for **data selection**. It is used in the **preliminary phase** of investigation but also at the end, in the **final data analysis**.
Sampling is used because obtaining and processing the **entire dataset** of interest is **too expensive or time consuming** : it works well if the sample is **representative** (= has approximately the same property) of the entire original data.

- **Simple Random Sampling** = equal probability of selecting any particular item

- **Sampling without replacement** = as each item is selected, it is removed from the population
- **Sampling with replacement** = objects are not removed and can be picked again (boostrap!)
- **Stratified sampling** = split data into several **homogeneous** partitions, then draw random samples from partitions

13.3 Aggregation

Combining two or more attributes (or objects) into a **single attribute** (or object) . This helps to **reduce** the number of attributes or objects . Aggregation is also useful to introduce a **change of scale** : cities are aggregated into region, states ,countries ...for example.
Aggregated data is useful because it tends to be more **stable** , have less **variability**.

13.4 Feature Creation

Create **new** attributes that can capture **important informations** in a dataset ,much more efficiently than the original attributes (e.g.: given an address create longitude and latitude attributes). The main operations that can be done to create new features:

- **Feature extraction**
- **Mapping data to a new space**
- **Feature construction** (combining features)

13.5 Discretization

Attributes can be :

- **Nominal** : values from **unordered set**.
- **Ordinal** : values from **ordered set**.

- **Continuous**: real numbers.

The process of **discretization** consists of dividing **continuous** attributes in discrete intervals (some classification algorithms only accept **categorical** attributes!). This also reduces the data size and prepares data for future data analysis.
It can be done in two ways:

- **Supervised**
  Attributes are discretized using **class information**. This methods tries to minimize the loss of information about the class. Used for example in IR Classifier for classification rules.

- **Unsupervised**
  Attributes are discretized solely based **on their values**.

13.6 Feature Selection (Dimensionality Reduction)

This step is required to avoid the **curse of dimensionality** , save amount of time and money required by data mining algorithms ,allow data to be visualized more easily and may help to eliminate irrelevant features or reduce noise.
Features may be **redundant** (duplicate all or almost all the information contained in one or more attributes) or **irrelevant** (contain no information that is useful for the data mining task).

- PCA

- SVD

- Other supervised / non-supervised techniques ...

Different **approaches** can be used:

- **Brute force** : try all possible subset of features as input to data mining algorithm

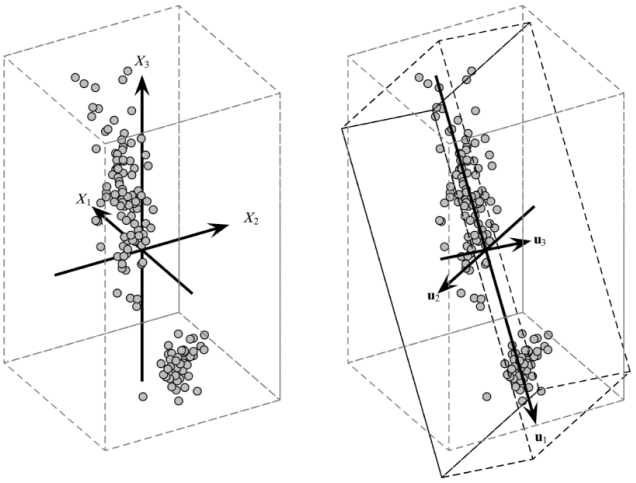
- **Embedded approaches** : feature selection occurs naturally as part of data mining algorithm (like lasso)

- **Filter approaches** : features are selected using a procedure that is **independent** from a specific data mining algorithm , for example using a **correlation matrix**.
- **Wrapper approaches**: use a data mining algorithm as a black box to find best subset of attributes (e.g.: genetic algorithm and decision tree to find best features of tree)

13.6.1 Filter approach 1 : PCA

PCA is the **principal component analysis** which seeks a **basis** that best captures the **variance** in the data.This method works only on **numerical data**. The direction with the **largest** projected variance is called **first principal component**. The **orthogonal** direction that captures the **second largest** projected variance is the second principal component, and so on.
Goal is to find, given N data vectors from n-dimensions , **k**< **n** orthogonal vectors:

1. **Normalize** input data
2. Compute k **orthonormal** vectors
3. Each input data vector is a **linear combination** of these components
4. The principal components are sorted in order of **decreasing significance**
5. Eliminate the **weak components** (low explained variance)



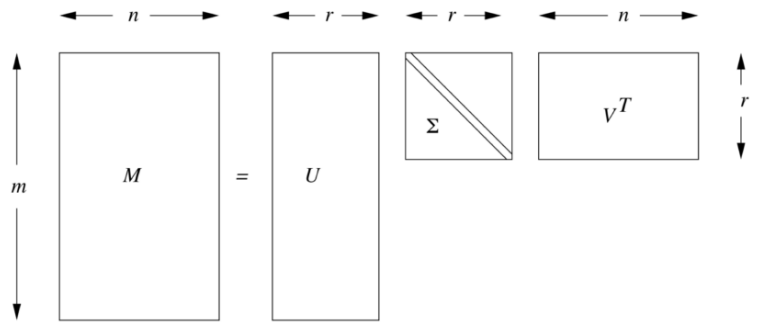
13.6.2 Filter approach 2 : SVD

PCA is a special case of a more general **matrix decomposition** approach **Singular Value Decomposition** : it is **always** possible to decompose matrix **M** into

$$M = U\Sigma V^T$$

where U, Σ, V are **unique**.

- **U**: is column orthonormal so that $UU^T = I$. This matrix represents the n samples using **r** new concepts/attributes
- Σ : represents the **strength** of each concept.$\Sigma_{i,j}$ are the **single values**,positive and sorted in decreasing order.
- **V** : is column orthonormal so that $VV^T = I$. This matrix contains m terms, r concepts.



An example:

	Matrix	Alien	Star Wars	Casablanca	Titanic
Joe	1	1	1	0	0
Jim	3	3	3	0	0
John	4	4	4	0	0
Jack	5	5	5	0	0
Jill	0	2	0	4	4
Jenny	0	0	0	5	5
Jane	0	1	0	2	2

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{bmatrix} =$$

M'

$$\begin{bmatrix} .13 & .02 & -.01 \\ .41 & .07 & -.03 \\ .55 & .09 & -.04 \\ .68 & .11 & -.05 \\ .15 & -.59 & .65 \\ .07 & -.73 & -.67 \\ .07 & -.29 & .32 \end{bmatrix} \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix} \begin{bmatrix} .56 & .59 & .56 & .09 & .09 \\ .12 & -.02 & .12 & -.69 & -.69 \\ .40 & -.80 & .40 & .09 & .09 \end{bmatrix}$$

U Σ V^{T}

As seen in the Σ matrix , which represents the **strength** of the concepts, the last column last row value 1.3 is the weakest and could be eliminated (eliminating also the corresponding column in U and row in V).

$$\begin{bmatrix} .13 & .02 \\ .41 & .07 \\ .55 & .09 \\ .68 & .11 \\ .15 & -.59 \\ .07 & -.73 \\ .07 & -.29 \end{bmatrix} \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \begin{bmatrix} .56 & .59 & .56 & .09 & .09 \\ .12 & -.02 & .12 & -.69 & -.69 \end{bmatrix}$$

$$= \begin{bmatrix} 0.93 & 0.95 & 0.93 & .014 & .014 \\ 2.93 & 2.99 & 2.93 & .000 & .000 \\ 3.92 & 4.01 & 3.92 & .026 & .026 \\ 4.84 & 4.96 & 4.84 & .040 & .040 \\ 0.37 & 1.21 & 0.37 & 4.04 & 4.04 \\ 0.35 & 0.65 & 0.35 & 4.87 & 4.87 \\ 0.16 & 0.57 & 0.16 & 1.98 & 1.98 \end{bmatrix}$$

How many singular values?

Retain enough singular values to make up 90% of the **energy** in Σ: the **sum of squares** of the **retained** singular values should be at least 90% of the energy of the singular values.

Example:

- $(12.4)^2 + (9.5)^2 + (1.3)^2 = 245.70$
- $(12.4)^2 + (9.5)^2 = 244.01 \approx 99\%$!
- $(12.4)^2 = 153.76 \approx 63\%$

13.6.3 Wrapper Approach

Focus on specific data mining algorithm and find the best subset of attributes for that specific algorithm . Example :find subset of features that maximizes DT performance. The algorithm is applied as a black box and uses a **search method** to find the best subset. Different approaches:

- **Backward feature elimination**: start training on n input features, then remove one input at the time and train on n-1 input.
  The input feature whose removal has produced the **smallest increase** in error rate is removed.
  Each iteration k produces a model trained on n-k features and an error rate e(k). Stops when maximum tolerated error rate is reached.

- **Forward feature creation**: inverse process of BFE.
- **Recursively feature creation** : repeatedly creates a model and keeps aside the **best** and **worst** performing feature at each iteration. It construct the next model using the **remaining features** and then **ranks them** based on the order of elimination.

13.6.4 Filter vs Wrapper

- Filter approaches focus on the **relevance of features** while wrapper measure the **usefulness** of a subset of features by actually training a model
- Filter approaches are faster since they do not require training (wrappers are computationally expensive!)
- Filter methods use **statistical methods** for evaluation while wrapper use **cross-validation**
- Filter methods **fail** to find the **best subset** in many occasions while wrapper can **always** provide the best subset
- Using the wrapper subset makes it more prone to overfitting!

14 Text Mining

Text mining deals with the discovery of **new** ,previously **unknown** ,**information** by extracting automatically information from different **unstructured textual documents**.
Text mining is used for:

- Classification
- Clustering
- Sentiment Analysis
- Summarization
- Relation/Concepts/Events extraction
- Topic modelling
- ...

The goal is to turn **text data** into high-quality information that :

- minimizes **human effort**
- supplies **knowledge** for optimal decision making

Text mining is strongly related to **text retrieval** which is an essential part of the task.

14.1 NLP

Natural language processing is a very important field in text mining. Natural language is designed for **efficient** human communication. This often results in **omitting** a lot of **common sense knowledge** which we assume the reader/-hearer possesses and results also in a lot of **ambiguities** which we assume the reader/hearer know how to resolve.
This makes NLP very difficult for many reasons :

- **Word-level ambiguity** : "design" can be noun or verb, "root" as many meanings...

- **Syntactic ambiguity** : "a man saw a boy with a telescope"
- **Presupposition** : "he has quit smoking" implies he smoked

An is as today still a field with many open problems.

14.2 Information Retrieval

Deals with the problem of locating **relevant documents** with respect to the **user input or preference** used for example in on-line library catalogs or document management systems.
Must deal with :

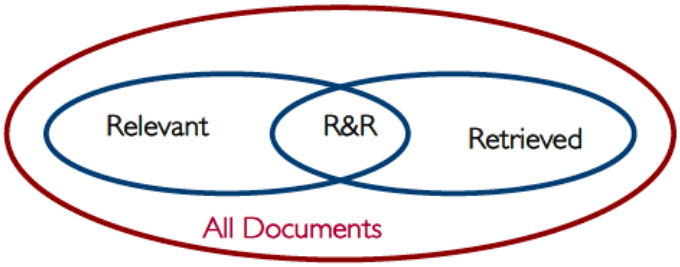
- Management of **unstructured documents**
- Approximate search
- Relevance

There are 2 modes of **text access**:

1. **Pull mode (Search Engine)**
   The user takes initiative and gets ad-hoc information
2. **Push mode (Recommender Systems)**
   The system takes initiative having good knowledge about a user's need.

Then there are 2 steps in **text retrieval**:

1. **Document Selection**
   This is **keyword-based** retrieval where a **query** defines a series of requisites and only documents that satisfy them are returned.
   E.g.: **Boolean Retrieval Method**
2. **Document Ranking**
   This is **similarity-based** retireval where documents are ranked on basis of relevance wrt to the user query : for each document a **degree or relevance** is computed wrt to the query.
   E.g.: **Vector Space Model**

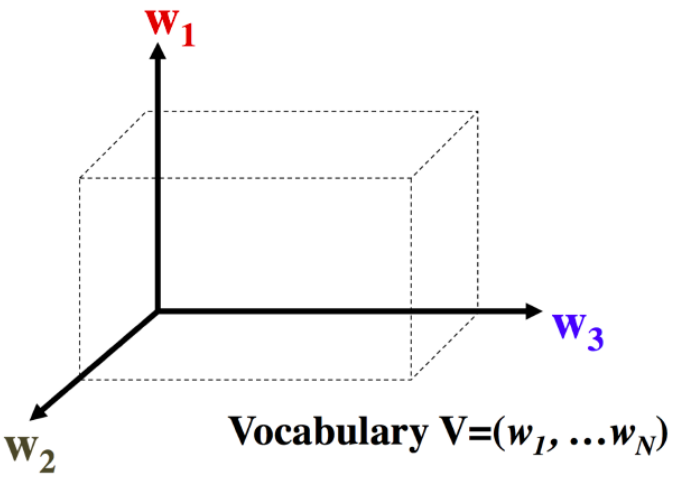


$$\text{precision} = \frac{|\{\text{Relevant}\} \cap \{\text{Retrieved}\}|}{\{\text{Retrieved}\}}$$

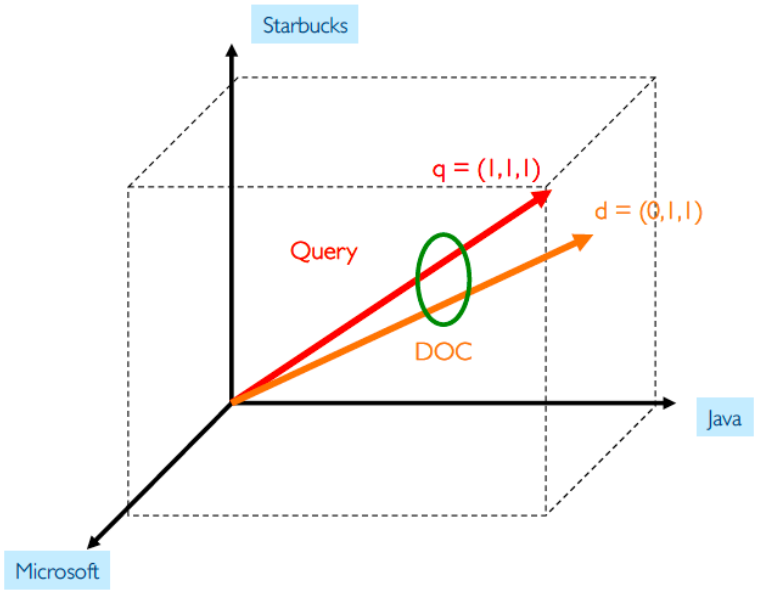
$$\text{recall} = \frac{|\{\text{Relevant}\} \cap \{\text{Retrieved}\}|}{\{\text{Relevant}\}}$$

$$\text{F-score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

A document and a query are represented as **vectors** in high-dimensional space corresponding to **all the keywords**. **Relevance** is measured with an appropriate similarity measure defined over the vector space.
How to select **keywords** to capture **basic concepts**?
How to assign weights to them?
How to measure similarity?



In the figure an example with a 3 word vocabulary that creates a 3D space and below an example of document (bag of words does not consider **order**) and query inside such a space.



The similarity is given by the **angle** that forms between the two vectors.

14.3 Ranking Documents

Given

- a query $q = q_1, ..., q_m$ where q_i is a word
- a document $d = d_1, ..., d_n$ where d_i is a word (bag of words model)
- **Ranking function** f(q,d) return a **real value**

A **good** ranking function should put relevant documents over non-relevant documents.The key challenge is how to measure the **likelihood** that document d is relevant to query q :

- **Similarity-based models** : Uses the vector space model (our focus!)

$$f(q, d) = similarity(q, d)$$

- **Probabilistic models**: Classic probabilistic models, language models or divergence from randomness models

$$d(q, d) = p(R = 1|d, q)$$

- **Probabilistic inference models**:

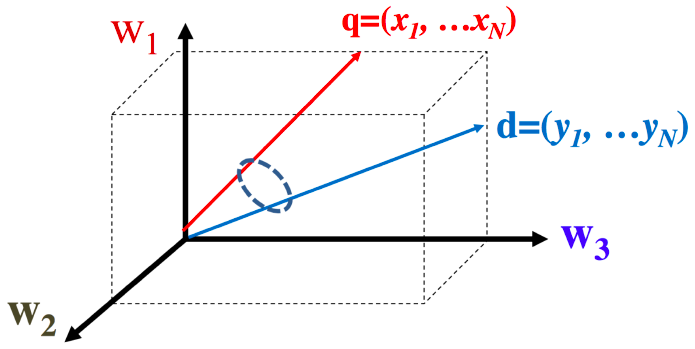
$$f(q, d) = p(d \implies q)$$

- **Axiomatic model**: $f(q, d)$ must satisfy a series of constraints.

14.3.1 1.Document Similarity Retrieval:VSM

A way to measure similarity between query and documents is using the dot product:

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



Each word is either 0 or 1 depending on if it is found in the document/query or not:

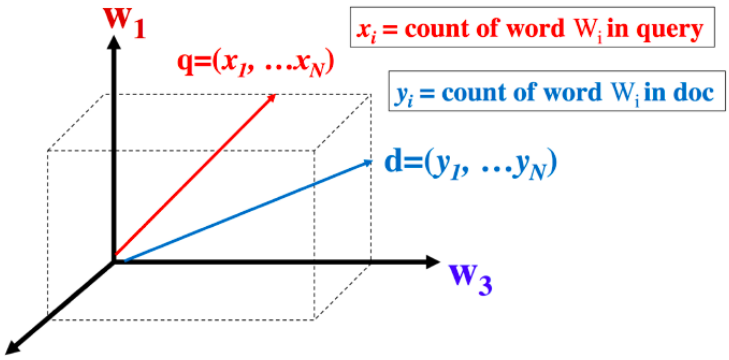
Query = "news about presidential campaign"

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 a **basic approach** which yields good results but can be improved : when searching for a query certain words are repeated more than once and deserve more credit or certain words have **bigger weight** than others (e.g.: presidential > about)

14.3.2 2.Document Similarity Retrieval : TFW

A solution to weighting common words more is used **Term Frequency Weighting**:



$q=(x_1, \ldots x_N)$ x_i = count of word W_i in query

$d=(y_1, \ldots y_N)$ y_i = count of word W_i in doc

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

An example :

d2 … news about organic food campaign… f(q,d2)=3

q= (1, 1, 1, 1, 0, …)
d2= (1, 1, 0, 1, 1, …)

d3 … news of presidential campaign … f(q,d3)=3

q= (1, 1, 1, 1, 0, …)
d3= (1, 0, 1, 1, 0, …)

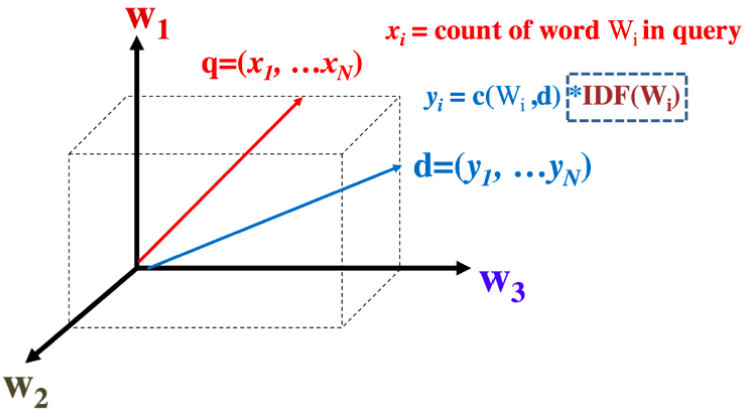
d4 … news of presidential campaign …
… presidential candidate … f(q,d4)=4!

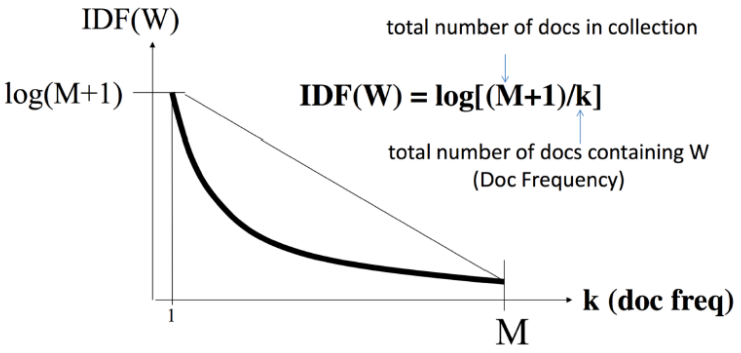
q= (1, 1, 1, 1, 0, …)
d4= (1, 0, 2, 1, 0, …)

This still does not solve that some words are more important than other (”presidential” > ”about”)

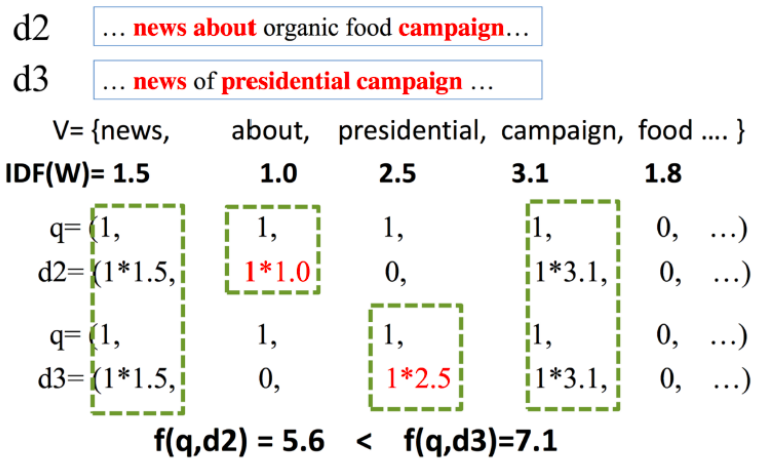
14.3.3 3.Document Similairty Retrieval : IDF

TFW can be extended using the **Inverse Document Frequency**

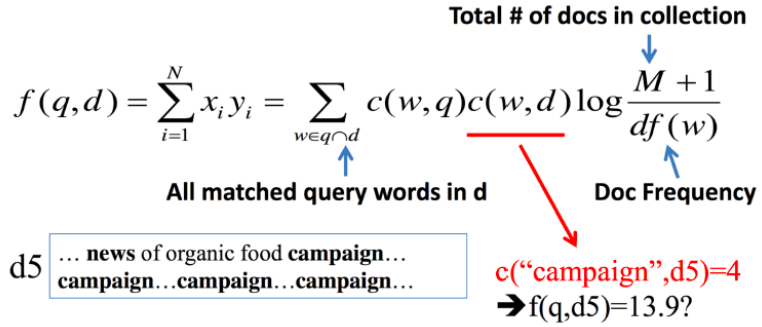




In this case "about" which is more common gets penalized wrt to "presidential" , a rarer word.



The more uncommon a word the higher the IDF score. Usually the denominator uses **k+1** to avoid 0 occurrence. Combining TF-IDF:

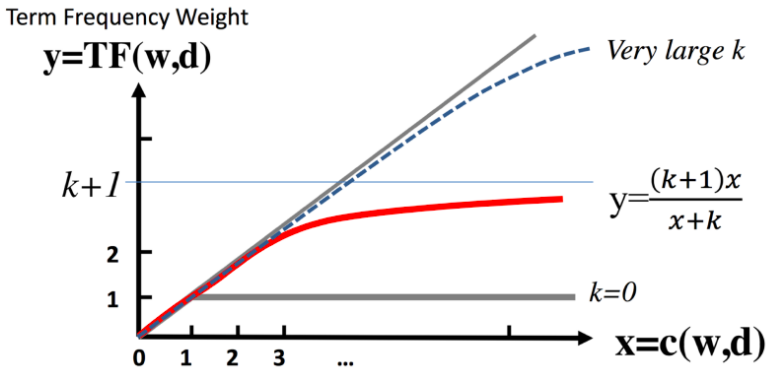


The term in red is what causes the last issue: the document d5 has still a score that is too high! It should be rated less , since it contains campaign , which is not common , **many times** , but the document is actually not so important! This because it appears not often , but then it appears often in **one document**.

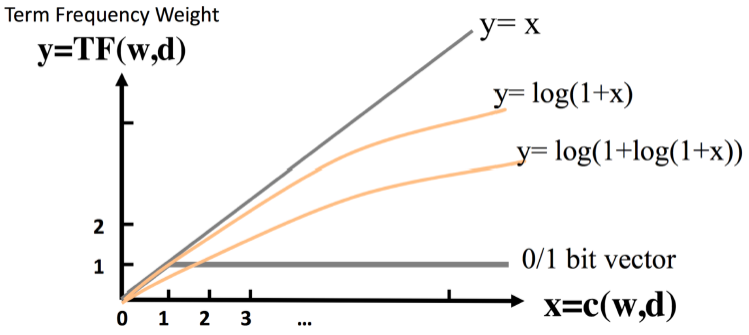
14.3.4 4.Document Similarity Retrieval : TFTransformations

This can be solved by two **transformations** that put an **upper limit** to the count of term frequency (modifies the count of terms in the document):

- **TFTransformation**



- **BM25**



A final model ,using **BM25** and **IDF** to rank documents :

$$f(q,d) = \sum_{i=1}^{N} x_i y_i = \sum_{w \in q \cap d} c(w,q) \frac{(k+1)c(w,d)}{c(w,d)+k} \log \frac{M+1}{df(w)}$$

14.3.5 5.Document Similarity Retrieval : Document Length

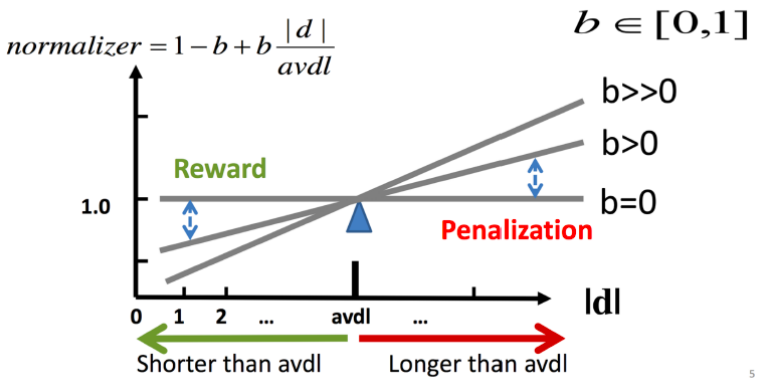
Documents need to be **normalized** in length. This is because longer documents have a better chance of matching the query than shorter ones and thus need to be **penalized** (avoid **over-penalization!**) A document is long because:

- it has **more word** → should be **penalized more**

- it has **more content** → should be **penalized less**

The **pivoted doc normalizer**:

- uses **average** doc length as **pivot**
- **1** if length $|d| = avgdl$



14.3.6 6.Document Similarity Retrieval : State of Art

- Pivoted Length Normalization VSM [Singhal et al 96]

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

- BM25/Okapi [Robertson & Walker 94]

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

14.4 From text to numerical vectors

Text is processed using **tokens** ,and textbfkeywords are isolated and identified with:

- **Stop word elimination**
  Stop words are elements that are considered **uninteresting** wrt to the retrieval and thus eliminated (e.g. : "a" ,"the", "always"...)
- **Word Stemming**
  Different words share **common prefix** which can replace whole words . E.g.: "computer" , "computing" ,"computerize" are **replace with** "comput"

Another task to deal with is **dimensionality reduction**.Working on high dimensional space with **many many keywords**:

- **computationally expensive**
- problems with **synonymy** and **polysemy**

Dimensionality reduction techniques include :

- Probabilistic semantic indexing
- Locality preserving index
- **Latent Semantic Indexing**
  Let x_i be vectors representing documents and X the set of all documents :

  $$\overrightarrow{x_i}, ..., \overrightarrow{x_n} \in R^m \quad X = [\overrightarrow{x_1}, ..., \overrightarrow{x_n}]$$

  Use **SVD** to reduce the size of frequency table :

  $$X = U\Sigma V^T$$

  Then approximate $X \approx X_k$ that is obtained from the **first k vectors of U**.

14.5 Text Classification

Solves the problem of labeling **automatically** text documents on the basis of :

- Topic
- Style
- Purpose

Usual classification techniques can be used to learn from a training set of **manually labeled** documents.
Features could be : **keywords**.

14.5.1 Similarity based text classifiers

Exploits information retrieval and **k-nearest neighbour**:

- for a new document to classify find the **k** most similar ones
- documents are classified on the basis of class distribution among the k documents retrieved using **majority vote**
- tuning k is **very important for good results**

Limitations are due to **space overhead** (save documents) and **time overhead** (retrieve similar documents)

14.5.2 Dimensionality reduction in text classifiers

Similar to what is done in VSM to reduce the number of features to represent text : they usually use **distribution of keywords** among the **whole database**.
In classification also **correlation** between **keywords - class label** must be taken into account : rare keywords have a high TF-IDF but might be uniformly distributed among classes.
After the dimensionality reduction step a classifier can be applied (SVM ,Naives Bayes..)

14.5.3 Example with Naives Bayes Text Classifier

- Document categories $\{C_1, ..., C_n\}$
- Document to classify D
- Probabilistic model

$$P(C_i|D) = \frac{P(D|C_i)P(C_i)}{P(D)}$$

- Choose class $C^* = argmax_c P(C|D) = argmax_C P(D|C)P(C)$

Features can be defined as **words in the document**:

$$P(D|C) = P(a_1 = w_{j_1}, a_2 = w_{j_2}, \cdots, a_n = w_{j_n}|C)$$

Example

H={like,dislike}
D= "Our approach to representing arbitrary text documents is disturbingly simple"

$$P(D|Like) = P(a_1 = \text{our}, a_2 = \text{approach}, \cdots, a_10 = \text{simple}|Like)$$

- Assumptions
  - Keywords distributions are inter-independent
  - Keywords distributions are order-independent

$$P(D|C) = \prod_{i=1}^{n} P(w_{j_i}|C)$$

$$P(D|Like) = P(\text{our}|Like)P(\text{approach}|Like)\cdots P(\text{simple}|Like)$$

- How to compute probabilities?
  - Simply counting the occurrences may lead to wrong results when probabilities are small
- M-estimate approach adapted for text:

$$P(w_k|C) = \frac{N_{c,k} + 1}{N_c + |\text{Vocabulary}|}$$

  - N_c is the whole number of word positions in documents of class C
  - $N_{c,k}$ is the number of occurrences of w_k in documents of class C
  - |Vocabulary| is the number of distinct words in training set
  - Uniform priors are assumed

- Final classification is performed as

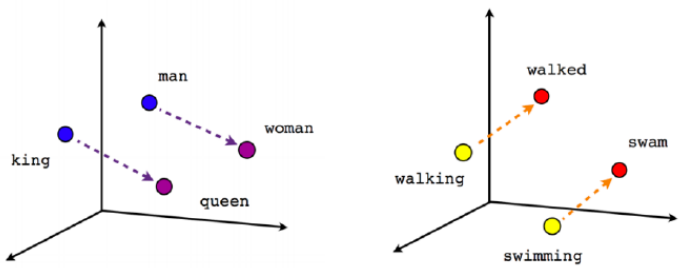
$$C^* = \arg\max_C P(C) \prod_{i=1}^{n} P(w_{j_i}|C)$$

- Despite its simplicity Naïve Bayes classifiers works very well in practice

- Applications
  - Newsgroup post classification
  - NewsWeeder (news recommender)

14.6 Word Embeddings

Learned representation for text where words with **similar meanings** have **similar representations**. It a set of methods that represent **individual words** as **real-valued vectors** (often up to 10.000 dimensions) in a predefined vector space: each vector is mapped to one vector and the vector values are **learned** in a way similar to NN.
Words and phrases from the vocabulary are **mapped** to vectors of real numbers.



Male-Female Verbe Tense

vector[Queen] = vector[King] - vector[Man] + vector[Woman]

This differentiates from the **word of bags model**:

- uses **one hot encoding**

- **one bit position** in a huge vector
- no context information

Word Embeddings :

- each words is a **point** in continuous space
- a word is represented by a vector of fixed number of dimensions (generally 300)
- Generated from a **huge corpus** using **unsupervised methods**
- Dimensions are projections along different axis

Word embeddings are very useful for :

- **word similarity**
- **machine translation**
- **part of speech and name-entity recognition**
- **relation extraction**
- **sentiment analysis** = usually uses bag of words
- **co-reference resolution** = chaining entity mentions across multiple documents :find and unify all contexts where mention occurs?
- **clustering** = words in same class occur naturally in similar contexts and have similar embeddings (k-means....)
- **Semantic Doc Analysis** = build word distributions for various topics...

Word embeddings can be done using :

- Latent semantic indexing (LSI)
- Fastext
- Word2Vec

14.6.1 Word2Vec

Uses a NN with **single hidden layer** to ,given a specific word in the middle of a sentence (**input word**) predict the **probability** for every word in our vocabulary of being a **nearby word** that we chose.
The **hidden layer** is just a **representation** of the input word while the output vector is a **probability distribution**.