INFOH423 - Data Mining

Jose Antonio Lorencio Abril

Fall 2022

Professor: Mahmoud Sakr

Student e-mail: jose.lorencio.abril@ulb.be

This is a summary of the course *Data Mining*, taught at the Université Libre de Bruxelles by Professor Mahmoud Sakr in the academic year 22/23. Most of the content of this document is adapted from the course notes by Sakr, [3], and the basic bibliographic source of the course, the book of Aggarwal, [1], so I won't be citing it all the time. Other references will be provided when used.

Contents

List of Figures

List of Tables

List of Algorithms

Part I
Introduction

1 What is data mining?

Even though there is not a formal-accepted-by-all definition of data mining, most of them are in agreement that data mining is a field of study that focuses on the collection, cleaning, processing, analysis and gaining of useful insights from the data that we have access to.
As wider as these topics are, is also the data mining domain, which is nowadays a really hot topic both in academia and industry.
In academia, there exist many goals to achieve in this field:

- Developing new models.
- Developing new ways to deal with real world data.
- Understading what some complex models are actually doing.
- Developing the mathematical tools to be able to better describe the models that are made by computer scientist in a less formalized way (it is not rare that scientific models come before their mathematical formalization[1].
- Using of data mining to gain a better understanding of medical, biological, chemical, economic,... data than we are able to gain using just traditional statistic techniques.

In the industry, the focus is mainly in two fields (disregarding research companies, which would have similar objectives as academia):

- Using well known models to understand what happened in the past.
- Using well known models to try to predict what will happen in the future.

These usages are made with the aim of improving the business processes and, ultimately, maximize profit.

1.1 Why is data mining important today, if it was not yesterday?

Because the computing power has increased enormously, so now we are able to run algorithms that enable us to train models in a decent time, and this task was practically undoable a few years ago.
Not only that, but the data technology is rapidly evolving, too. We produce more data and store more data, so... we have more data! This data is potential knowledge and we know many tools to get this knowledge from it.
In fact, the amount of data is so vast, that not only we have to develop techniques to analyze data, but also to manage huge amounts of data.

2 The Data Mining Process

The data mining process is a pipeline constructed around the basic steps:

1. **Data collection**: obtention of data from real world sources.
2. **Feature extraction** and **data cleaning**: among all data retrieved from the real world, we have to select those characteristics or **features** that are relevant for our purposes (feature extraction) and to decide what to do with mistaken/noisy/lost data (data cleaning). Also, as we might be collecting data from different sources, it is important to decide how to aggregate the data into a unified format for later processing.

[1] For example, for Newton, differential calculus was just a tool he developed to be able to better understand some physical processes that were kind of obvious to him. Afterwards, it became one of the widest branches of mathematics and led to other mathematical tools, theories and developments in many other fields.

index	Name	Age	Height	Gender
1	Josh	23	1.77	M
2	Mary	28	1.62	F
3	Larry	12	1.58	M

Table 1: Records and multidimensional data set example

3. **Analytical processing** and **algorithms**: we now must develop suitable methods for analyzing our data. That is, we should decide what mathematical models to use to describe our data, and what algorithms to use in order to *train* the desired model[2].

4. After these steps, we would enter a phase of analyzing the obtained results to obtain knowledge from data, as well as a iterative procedure, in which we could return to any previous step and try to improve different parts of the process[3].

3 Data Types

Not all data have have the same nature nor characteristic, so it is important to understand the differences between them and what techniques are applicable to what types of data.
We can characterize data in different levels of detail, and the most basic classification would distinguish between *nondependency-oriented* data and *dependency-oriented* data:

- **Nondependency-oriented data**: simple data types with no specified dependencies between the data items or the attributes.
- **Dependency-oriented data**: implicit or explicit relationships may exist between data items.

Dependency-oriented data are normally more complex to study because of the need to study not only the data itself, but also the relationships between different data items.
We will now define different subtypes of data that we can find.

3.1 Nondependency-oriented data

The term nondependency-oriented data is interchangeable with the term **multidimensional data:**

Definition 3.1. We will call **source space**, $\mathcal{S}$, to the set of all possible values that our data can take. This set does not have neccessarily to take any particular form.
If $\mathcal{S}$ is a product space, then each component is called a **feature**.

Example 3.1. If we are measuring the name, age, height and gender of the students of a school, then we will have

$$\mathcal{S} = \mathcal{T} \times \mathbb{N} \cap [0, 150] \times (0, 3) \times \{M, F\},$$

where $\mathcal{T}$ represents the set of possible names and $\{M, F\}$ are the two possible values of the gender.

Definition 3.2. A **record** (*data point, instance, tuple*) is just a point $X = \left(x^i\right)_{i=1}^d \in \mathcal{S}$ that we measure and store in some form.

Example 3.2. Following the previous example, some records of $\mathcal{S}$ are represented in the following table: Note that we added an index column, because it is a common practice.

[2]We will see what is specifically the meaning of *train*, but in the meanwhile we can think of it as how we make a general model adapt to our particular data.
[3]Usually, data is constantly updating, so at least it will necessary to produce checks of correctness and updates.

Definition 3.3. A **multidimensional data set**, $\mathcal{D}$, is a set of n records, $\{X_j\}_{j=1}^{n}$.

Example 3.3. Table 3.2 represents an example of a dataset, too.

As we can see, Table 3.2 contains attributes of different types (very obvious from the definition of the source space $\mathcal{S}$). Thus, we have to take into account also the type of each attributes of our data:

- **Quantitative multidimensional data**: numerical data features, as age or height. If a data set is wholy compound of this kind of features, it is said to be a quantitative multidimensional data.

  This type of data is the easiest to analyze, as mathematical tools are directly applicable and most algorithms are developed assuming this type of data. For this reason, it is common to try to transform all non-quantitative data into quantitative data.

- **Categorical and Mixed Attribute data**: a **categorical** feature is such that it can only take values among a finite set (unordered) of options, as the gender.

  If we encounter a data set compound of categorial data, we would say it is a categorical multidimensional data.

  A dataset with both quantitative and categorical features is called a mixed multidimensional data.

- **Binary and Set data**: binary data take values in $\{0, 1\}$ and it can be considered a special case of both numerical data (obviously) and categorical data (as if have a categorical feature which can only take two values, then it is easy to map these values to the set $\{0, 1\}$). For example, the gender is an obvious case of binary data.

  Moreover, binary data can be seen as setwise data, where 1 indicates that the instance is in the set, while 0 indicates it is not.

- **Text data**: usually a string, such as the name.

3.2 Dependency-oriented data

As outlined before, we can find implicit or explicit dependencies between instances:

- **Implicit dependencies**: the dependencies between instancies are not explicitly specified but are known to exist. For example, if we measure the number of students in the library every 5 minutes, we would find that different instances are related via the temporal dimension, so measures with little time delay between them would be similar.

- **Explicit dependencies**: this term usually refers to graph or network data, in which edges represents relationships between nodes.

As before, let deepen a bit in some types of this kind of data:

- **Time-Series data**: it contains data that are generated by continuous measurement over time. This means that our source space has a temporal component. This dependency is implicit.

  Formally, a **time series of length n and dimensionality d** contains d numeric features at each of n time stamps $t_1, ..., t_n$. Each time stamp contains a component for each of the d series. Therefore, the set of values received at time stamp t_i es $Y_i = \left(y_i^1, ..., y_i^d\right)$.

- **Discrete sequences**: these are the categorical analog of time-series data. We will now have categorical or text features along the temporal dimension. This dependency is implicit.

- **Spatial data**: in this type of data, the dependance of the instances is given by their proximity in space. For example, if we measure the temperature in a room per cm^3, we will find that points that are nearby show more similar temperature than points that are far away from each other. This dependency is implicit.

- **Spatiotemporal data**: this data captures both spatial and temporal dimension, so we have to deal with both relationships. As before, this dependency is implicit.

- **Network and graph data**: now, data values may correspond to nodes in the network, and the relationships between them would correspond to the edges between the nodes.

  Formally, a **network** is a pair $G = (N, E)$ where N is a set of nodes and $E \subset N \times N$ is a set of edges, that represent the relationships between the nodes. There can be attributes associated to both nodes or edges.

  Edges may be directed or indirected, depending on wether the link is bidirectional or not.

Part II
Classification

A **classification problem** consists in learning the structure of a dataset of examples, already partitioned into groups, referred as **class**. This learning is typically achieved with a **model**, which is used to estimate the **class labels** of unseen data examples with unknown labels. Thus, one of the inputs to the classification problem is the example dataset with known labels, $\mathcal{D}$, called **training data**, while the unseen data points to be classified are the **test data**. The model learnt is referred to as **training model**. The algorithm used to create the model is the **learner**.
The output of the classification algorithm can be of two types:

- **Label prediction**: a label is predicted for each test instance.
- **Numerical score**: the learner assigns a score to each instance-label possible combination. This score measures the propensity of the instance to belong to a particular class.

4 Decision Trees

Decision trees are a classification methodology, which uses a tree structure to partition the feature space. Each node of the tree represents a decision to make according to the data, called the **split criterion**, and is a condition on one or more features variables in the training data.
The goal is to identify a split criterion such that the level of *mixing* of the class variables in each branch of the tree is reduced as much as possible.
The splits can be **univariate**, if they use a single attribute in the condition, or **multivariate**, if more than one attribute are used in the condition.
The nodes can be of two types:

- **Internal node**: each internal node represents a partition of the space according to a certain condition.
- **Leaf node**: they are labeled with the dominant class of the remaining partition of the training set at that node.

The general algorithm for constructing a decision tree is as follows:

Algorithm 1 GenericDecisionTree(Dataset: D)

```
begin
        Create root node containing D;
        repeat
                Select an eligible node in tree;
                Split the selected node into two or more nodes based on the split criterion;
        until no more eligible nodes for split;
        Prune overfitting nodes from tree;
        Label each leaf node with its dominant class;
end
```

The red lines indicate what changes accross the different algorithms to produce a specific decision tree.

4.1 Split criteria

The split criterion aims to maximize the separation of the different classes among the children nodes. Its design depends on the attributes of the data:

- Binary attributes: produce a binary tree.
- Categorical attribute: there several approaches:

  - r-way split: we split the branch in as many branches as distinct values of the attribute.
  - binary split: testing each of the $2^r - 1$ groupings of categorical attributes, and selecting the best one.

- Numeric attribute: we have, again, several possibilities:
  - If it contains a small number r of ordered values, we can treat it as a categorial attribute and apply r-way split.
  - For continuous numeric attributes, the split is performed using a binary condition, like $x \leq a$ for a certain contant a.

These methods require to determine the best split among a set of different splits, so we need a way to measure which one is better than other.
For this end, we are using the **entropy**.

4.1.1 Entropy

Definition 4.1. Let p_j be the fraction of data points belonging to the class j for the attribute value v_j. Then, the **class-based entropy**, $E(v_i)$, **for the attribute** value v_i is

$$E(v_i) = -\sum_{j=1}^{k} p_j \log_2(p_j).$$

Remark 4.1. When $p_j = 0$, it is assumed that $p_j \log_2(p_j) = 0$.

Remark 4.2. $E(v_i) \in [0, \log_2 k]$.

Remark 4.3. Higher values of the entropy imply greater *mixing* of different classes, while a value of 0 implies perfect separation.

Definition 4.2. The **overal entropy of an attribute**, E, is defined as the weighted average over the r different attribute values:

$$E = \sum_{i=1}^{r} \frac{n_i}{n} E(v_i),$$

where n_i is the frequency of attribute value v_i.

The entropy is used in the ID3 algorithm for constructing decision trees.
The overall entropy for an r-way split of set S into sets $S_1, ..., S_r$ may be computed as the weighted average of the entropy values of each S_i, being its weigth $|S_i|$. This is called the **entropy-split**:

$$Entropy - Split(S \implies S_1, ..., S_r) = \sum_{k=1}^{r} \frac{|S_k|}{|S|} E(v_i|S_k).$$

In relation to this, the **information gain** is defined as the reduction of entropy due to the split:

$$IG(S \implies S_1, ..., S_r) = E(S) - Entropy - Split(S \implies S_1, ..., S_r).$$

Note that lower values of the entropy-split and higher values of the information gain are more desirable.
Sometimes, there are attributes with lots of distinct values, so using them to split the data reduces the entropy a lot, but are not very useful for prediction[4]. To account for this, we can divide the overall information gain with the normalization factor

$$-\sum_{i=1}^{r} \frac{|S_i|}{|S|} \log_2 \left(\frac{|S_i|}{|S|} \right),$$

which helps adjusting for the varying number of categorical values.

outlook	temperature	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

Table 2: The weather data. Source: [4].

Example 4.1. Entropy of the dataset 'Weather data'
Consider the following dataset:
If the class attribute is *play*, what is the entropy of this source?

$$\begin{aligned} E\left(play\right) =& - p_{yes} \log_2 \left(p_{yes}\right) - p_{no} \log_2 \left(p_{no}\right) \\ =& - \frac{9}{14} \log_2 \left(\frac{9}{14}\right) - \frac{5}{14} \log_2 \left(\frac{5}{14}\right) \\ =& 0.94. \end{aligned}$$

What if the Entropy-Split of the attribute *humidity*?

$$\begin{aligned} ES\left(S \implies S_{hum=high}, S_{hum=normal}\right) =& \frac{|S_{hum=high}|}{|S|} E\left(play|hum = high\right) + \frac{|S_{hum=normal}|}{|S|} E\left(play|hum = normal\right) \\ =& \frac{7}{14}\left[-\frac{3}{7}\log_2\left(\frac{3}{7}\right) - \frac{4}{7}\log_2\left(\frac{4}{7}\right)\right] + \frac{7}{14}\left[-\frac{6}{7}\log_2\left(\frac{6}{7}\right) - \frac{1}{7}\log_2\left(\frac{1}{7}\right)\right] \\ =& 0.7885. \end{aligned}$$

And the information gain?

$$\begin{aligned} IG\left(S \implies S_{hum=high}, S_{hum=normal}\right) =& E\left(play\right) - ES\left(S \implies S_{hum=high}, S_{hum=normal}\right) \\ =& 0.94 - 0.79 = 0.15. \end{aligned}$$

4.2 ID3 Tree Induction Algorithm

ID3 is an algorithm to construct decision trees, in which the split criterion is the maximization of the information gain and the prunning strategy is to stop if all the records in a node are of the same class. The algorithm is more detailed in Algorithm 2.

[4] Think, for example, in an ID. See Subsubsection 4.2.1.

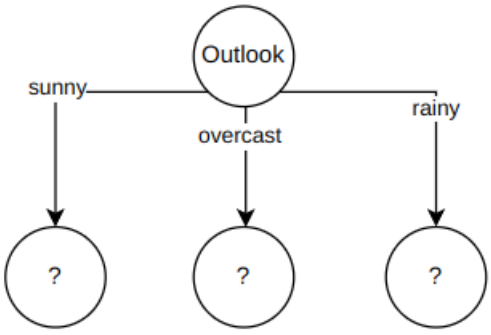
Algorithm 2 ID3(I, O, T) : Decision Tree

```
# I is the set of input attributes
# O is the output attribute
# T is a set of training data

if (T is empty) then
        return a single node with value "Failure"

if (all records  in T have the same value for O) then
        return a single node with that value

if (I is empty) then
        return a single node with the most frequent value of O in T

# else
compute IG for each attribute in I using data in T

Let X = argmin{IG(attr) for attr in I}
Let {x_j for j=1,...,m} be the values in X
Let {T_j for j=1,...,m} be the subsets of T when partitioned

return a tree with:
        root node labelled X
        arcs labelled x_1,...,x_m
        connected to
        ID3(I-{X}, O, T_1),...,ID3(I-{X}, O, T_m)
```

Example 4.2. Compute the decision tree of the data in Table 3.2 using ID3 algorithm.
Step 1: Root
We start by computing IG for each attribute. To simplify notation, let $P = play$, $O = outlook$, $T = temperature$, $H = humidity$ and $W = windy$.
We already know that $E(P) = 0.94$ and $IG(S \implies S_{H=high}, S_{H=normal}) = 0.15$. Let's compute the rest of the values:

$$ES(S \implies S_{O=sunny}, S_{O=overcast}, S_{O=rainy}) = -\frac{5}{14}\left(\frac{3}{5}\log\frac{3}{5} + \frac{2}{5}\log\frac{2}{5}\right)\cdot 2 - 0$$
$$=0.69.$$

Which implies that $IG(S \implies S_{O=sunny}, S_{O=overcast}, S_{O=rainy}) = 0.25$.
Repeating this process with temerature, we get $IG(S \implies S_{T=hot}, S_{T=mild}, S_{T=cold}) = 0.03$ and with windy, we get $IG(S \implies S_{W=True}, S_{W=False}) = 0.05$.
This means that we label the root node with $X = O$ and we create three arcs, each of them with one of the values from O. So we have the following Tree:



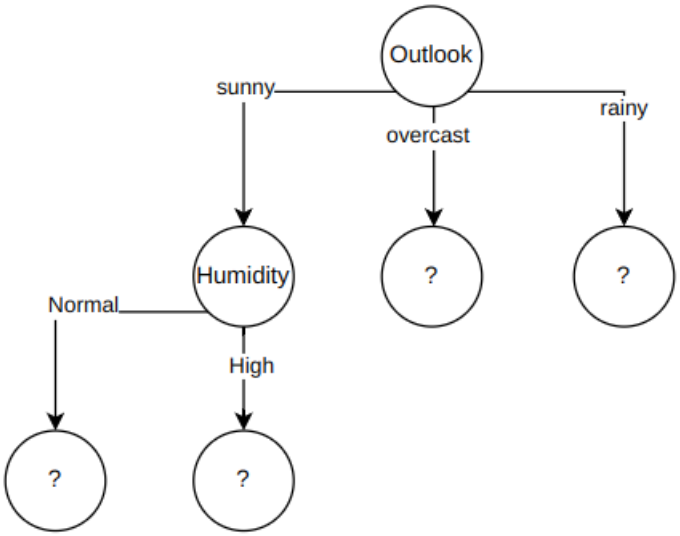
Step 2: Outlook=sunny
Now, we are going to do the same thing, restricting ourselves to the records for which *Outlook=sunny*. Now, we have to recompute the entropy and the gain for each of the rest of the attributes.
Let's start with the entropy:

$$E\left(P|O=sunny\right) = -\left[\frac{3}{5}\log\frac{3}{5}+\frac{2}{3}\log\frac{2}{5}\right] = 0.97.$$

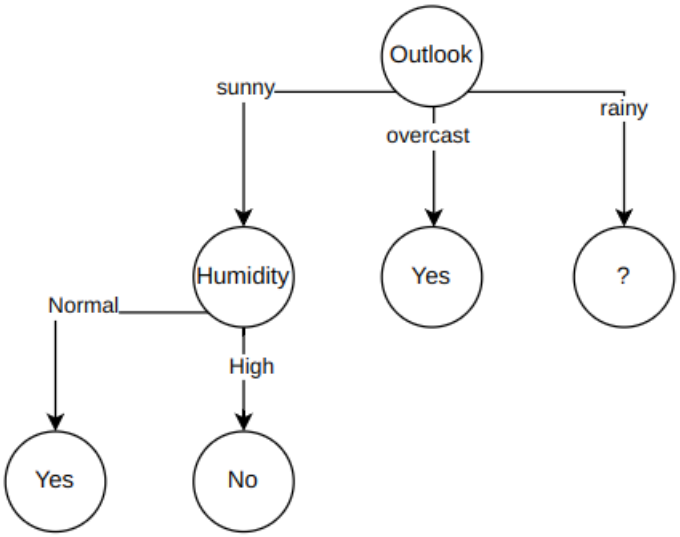
Now, the Information Gain:

$$IG\left(S_{O=sunny} \implies S_{O=sunny,H=high}, S_{O=sunny,H=normal}\right) =$$
$$\frac{3}{5}E\left(P \wedge O=sunny|H=high\right)+\frac{2}{5}E\left(P \wedge O=sunny|H=normal\right) =$$
$$-\frac{3}{5}\left[\frac{3}{3}\log\frac{3}{3}+0\right]-\frac{2}{5}\left[\frac{2}{2}\log\frac{2}{2}+0\right] = 0.$$

Which means that $IG\left(S_{O=sunny} \implies S_{O=sunny,H=high}, S_{O=sunny,H=normal}\right) = 0.97$. As this cannot be improved, we can savely not compute the rest of the values. This way, the Tree will now look as follows:



Step 3: Outlook=Sunny, Humidity=Normal
Note that we are proceeding heightwise, but doing this breadthwise is also possible.
This time, all the values for P are Yes, so we enter the third if of the algorithm and label the node as Yes.
Step 4: Outlook=Sunny, Humidity=High
Same, now $P = No$.
Step 5: Outlook=Overcast
Same, now $P = Yes$.
So, now we have the following tree:



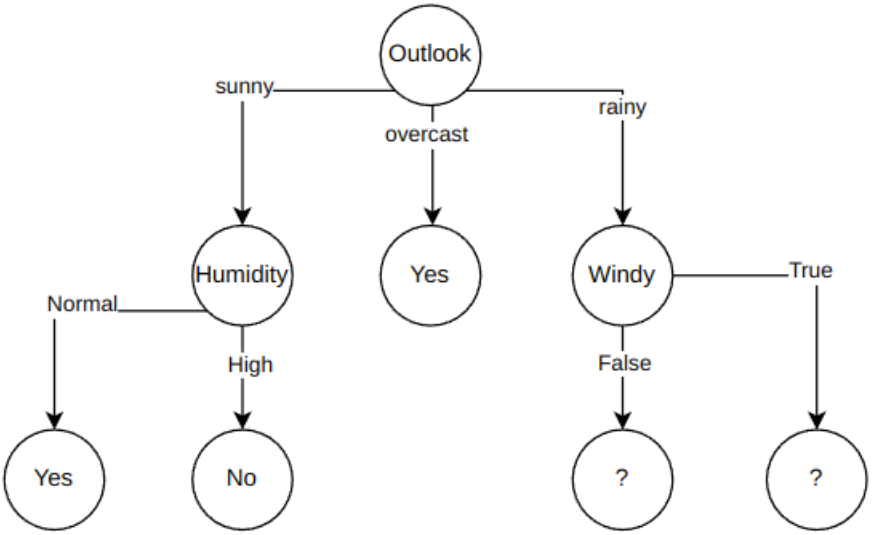
Step 6: Outlook=rainy

$$E\left(P|O = rainy\right) = 0.97$$

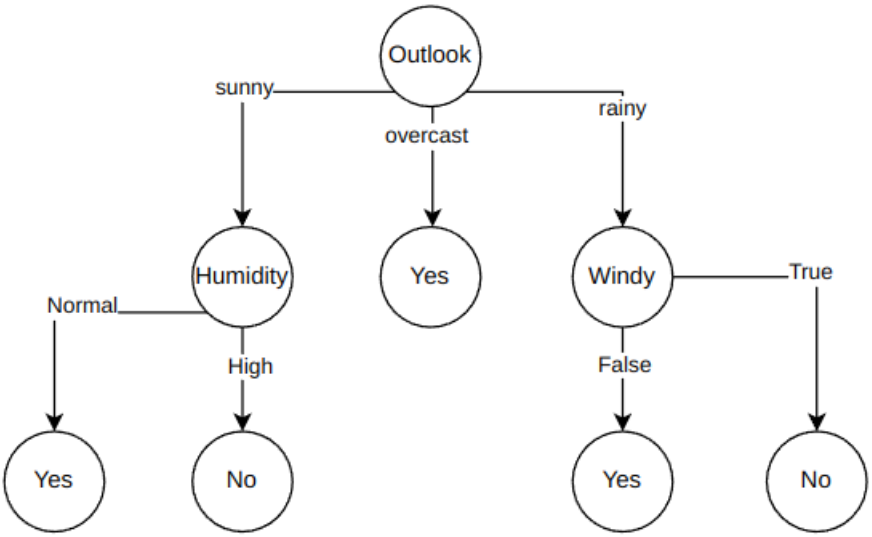
and

$$IG\left(S_{O=rainy} \implies S_{O=rainy,W=True}, S_{O=rainy,W=False}\right) = 0.97,$$

so, again, it is maximum and we can continue using it as label:



Step 7: Outlook=rainy, Windy=False
All records have $P = Yes$.
Step 8: Outlook=rainy, Windy=True
All record have $P = No$.
So, we have the tree



And as there are no more nodes to analyze, this is the final decision tree.

4.2.1 The problem of UID

In general, attributes that have very many values have very high gain, but can lead to useless decision trees. Quinlan suggest choosing the attribute with the highest

$$GainRatio\,(X,S) = \frac{Gain\,(X, S \implies S_1,...,S_r)}{Entropy\,(S)},$$

where X is the label attribute.
The GainRatio favores attributes with higher gain, and punishes attributes with high entropy (many values).

Example 4.3. Repeat the decision tree ID3 algorithm, but use GainRatio instead.
The same tree is obtained.

5 Bayesian classification

Probabilistic classifiers construct a model that quantifies the relationships between the feature variables and the target variable as a probability. We are going to study a well-known kind of probabilistic classifier, namely the **Bayesian classifier** or **Naive Bayes classifier**.

5.1 Naive Bayes classifier

The **Bayes classifier** is based on the Bayes' theorem for conditional probabilities.

Theorem 5.1. ***Bayes' Theorem***
If A and B are probabilistic events and $P\,(B) \neq 0$, then

$$P\,(A|B) = \frac{P\,(B|A)\,P\,(A)}{P\,(B)}.$$

Proof. On one side

$$P\,(A|B) = \frac{P\,(A \cap B)}{P\,(B)}.$$

On the other side, if $P(A) \neq 0$

$$P(B|A) = \frac{P(B \cap A)}{P(A)} \implies P(B \cap A) = P(B|A) \cdot P(A).$$

Now substituting this value in the previous equation, we get

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)},$$

which is the result we wanted to proof.
If $P(A) = 0$, then $P(A|B) = 0, \ \forall B$, so the formula also holds. □

This theorem quantifies the conditional probability of a random variable (the class variable), given known observations about another variables (the features). Let C be the class variable and X an unseen feature tuple. The goal of the method is to estimate

$$P(C = c|X = (a_1, ..., a_d)).$$

Let the random variables for the individual dimensions of X be denoted by $X = (x_1, ..., x_d)$, so we want to estimate

$$P(C = c|x_1 = a_1, ..., x_d = a_d) \overset{Bayes}{=} \frac{P(C = c) P(x_1 = a_1, ..., x_d = a_d|C = c)}{P(x_1 = a_1, ..., x_d = a_d)}.$$

Here, we will be interested in maximizing this value. Since the denominator is the same independently of the class c, then we can focus on maximizing the denumerator

$$P(C = c) P(x_1 = a_1, ..., x_d = a_d|C = c).$$

The value $P(C = c)$ is the **prior probability** of the class identifier c and can be estimated as the fraction of points in the data whose class is c. Thus, we now want to approximate the right factor. In the Naive Bayes approach, it is assumed that the feature values are independent of one another conditional on a fixed value of C. Then

$$P(x_1 = a_1, ..., x_d = a_d|C = c) = \prod_{j=1}^{d} P(x_j = a_j|C = c),$$

so

$$P(C = c) P(x_1 = a_1, ..., x_d = a_d|C = c) = P(C = c) \prod_{j=1}^{d} P(x_j = a_j|C = c).$$

And this terms are much easier to estimate: to estimate $P(x_j = a_j|C = c)$ we just need to take the fraction of training values with class c and compute which fraction of them verifies $x_j = a_j$. This is usually written as

$$P(x_j = a_j|C = c) = \frac{q(a_j, c)}{r(c)}.$$

Remark 5.1. When there are not enough training samples to produce reliable estimates, we can use **Laplacian smoothing** in which a small value α is added to the numerator and $\alpha \cdot m_j$ is added to the denominator, where m_j is the number of distinct values of the j^{th} attribute

$$P(x_j = a_j|C = c) = \frac{q(a_j, c) + \alpha}{r(c) + \alpha \cdot m_j}.$$

α is called the **Laplacian smoothing parameter**.

Remark 5.2. If a feature is continuous, then the likelihood is computed using a Gaussian distribution with mean μ and standard deviation σ:

$$P(x_j = a_j|C = c) = g(a_j, \mu_c, \sigma_c) = \frac{1}{\sqrt{2\pi}\sigma_c} e^{-\frac{(a_j - \mu_c)^2}{2\sigma_c^2}}.$$

This model is sometimes refererd to as the **Bernoully model for Bayes classification** when it is applied to categorical data with only two outcomes of each feature attribute.
In cases where more than two outcomes are possible for a feature variable, the model is referred to as the **generalized Bernoulli model**.

Example 5.1. With the following training data:

Age	Income	Student	Credit_Rating	Buys
≤ 30	high	no	fair	no
≤ 30	high	no	excellent	no
31..40	high	no	fair	yes
> 40	medium	no	fair	yes
> 40	low	yes	fair	yes
> 40	low	yes	excellent	no
31..40	low	yes	excellent	yes
≤ 30	medium	no	fair	no
≤ 30	low	yes	fair	yes
> 40	medium	yes	fair	yes
≤ 30	medium	yes	excellent	yes
31..40	medium	no	excellent	yes
31..40	high	yes	fair	yes
> 40	medium	no	excellent	no

Classify with the Naive Bayes model the unseen record

$$X = (\leq 30, medium, yes, fair)\,.$$

In this case, we have $C \in \{yes, no\}$, so

$$P\,(C = yes) = \frac{9}{14}, \qquad P\,(C = no) = \frac{5}{14}.$$

And for the feature variables, we have to compute $P\,(x_i = a_i|C = yes)$ and $P\,(x_i = a_i|C = no)$:

- Age ≤ 30:
$$P\,(\leq 30|yes) = \frac{2}{9}, \qquad P\,(\leq 30|no) = \frac{3}{5}.$$
- Income medium:
$$P\,(medium|yes) = \frac{4}{9}, \qquad P\,(medium|no) = \frac{2}{5}.$$
- Student yes:
$$P\,(yes|yes) = \frac{6}{9}, \qquad P\,(yes|no) = \frac{1}{5}.$$
- Credit_rating fair:
$$P\,(fair|yes) = \frac{6}{9}, \qquad P\,(fair|no) = \frac{2}{5}.$$

Then, we have

$$P\,(C = yes)\cdot P\,(\leq 30, medium, yes, fair|C = yes) = P\,(C = yes)\cdot\prod P\,(X_i|C = yes) = \frac{9}{14}\cdot\frac{2}{9}\frac{4}{9}\frac{6}{9}\frac{6}{9} = \frac{16}{567} \sim 0.028,$$

$$P\,(C = no)\cdot P\,(\leq 30, medium, yes, fair|C = no) = \frac{5}{14}\cdot\frac{3}{5}\frac{2}{5}\frac{1}{5}\frac{2}{5} = \frac{6}{875} \sim 0.007.$$

Thus, the model classifies X with class 'yes'.

Some final comments

The **advantages** of the Bayes Classifier is that it is easy to implement and can lead to fairly good results.
The **drawbacks** are that the main assumption of class conditional independence is not very trustable, which may lead to a loss of accuracy, because dependencies axist among variables. To deal with this issue there is a more complex model: **Bayesian Belief Networks**.

6 Model evaluation and selection

With a given dataset, we can train multiple classification models and each of them will behave differently, many times without an intuitive explanation of the differences observed. Thus, it becomes crucial to establish means of comparison between different models, so it is possible to choose between several models trained with the same data.
The simplest classification measure is the **accuracy**, which indicates the portion of well classified records. Nonetheless, if we compute the accuracy using the training data, we could be enhance overfitting to the training data, and maybe choose models that do not work well with unseen data. This is why it is usual to use a **validation test set** to compute comparison measures: the idea is that given a dataset, we can divide it into training data and test data. Then, models will be trained using the training data and evaluated with the test data, which has not been seen before. This makes the measures more reliable and the comparisons more fair.
There are several ways to extract training and test data from a dataset, which are detailed in 9.

6.1 Confusion Matrix

A confusion matrix is a visual and intuitive way to assess a classification algorithm. In its simplest form it is used to assess a binary classification model and it shows the following metrics:

- **True Positives**: count of records classified as True that are actually True.
- **False Positives**: count of records classified as True that were False in reality.
- **True Negatives**: count of records classified as False that were actually False.
- **False Negatives**: count of records classified as False that were True in reality.

In this case, the confusion matrix has the form:

	Predicted True	Predicted False
Actual True	TP	FN
Actual False	FP	TN

In the more general case in which we have a N-class classifier, each cell M_{ij} would have the count of records classified as class j that are of class i in reality:

	Predicted C_1	Predicted C_2	...	Predicted C_N
Actual C_1	TC_1	$FC_{2,1}$	...	$FC_{N,1}$
Actual C_2	$FC_{1,2}$	TC_2	...	$FC_{N,2}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
Actual C_N	$FC_{1,N}$	$FC_{2,N}$	...	TC_N

From these confusion matrices, we can derive some interesting measures:

- **Accuracy**: the accuracy can be computed from the matrix as

$$Acc = \frac{TP + TN}{All}.$$

- **Error rate**: the reverse of the accuracy:

$$ER = 1 - Acc = \frac{FP + FN}{All}.$$

- **Class imbalance measures**: sometimes, one class appears in a little amount of records (this is specially common in disease detection, for example), and is usually the case to have the focus on being able to correctly detect this rare cases.

  Example 6.1. As an example, imagine we are trying to detect fraudulent transactions in a banking context. Out of thousands of transactions, only a very few amount would be fraudulent. Say there are 1M records for training and only 100 are known to be fraudulent. If we predict all records as OK, we would get $\frac{999\ 900}{10^6} = 99.99\%$ accuracy, but we will not detect any fraud! So the accuracy is stunning but the model is useless.

  For example like this the following measures have a motive:

  - **Sensitivity**: focuses on correctly detecting True outcomes

    $$Sens = \frac{TP}{Actual\ True}.$$

  - **Specificity**: focuses on correctly detecting False outcomes

    $$Spec = \frac{TN}{Actual\ False}.$$

- **Precision**: how many records labeled as True are actually True

  $$Prec = \frac{TP}{TP + FP}.$$

- **Recall**: a synonym of sensitivity.

- **F-score**: the harmonic mean of the precision and recall

  $$F = \frac{2 \cdot precision \cdot recall}{precision + recall}.$$

- $\boldsymbol{F_\beta}$: weighted measure of precision and recall. Assigns β times as much weight to recall as to precision

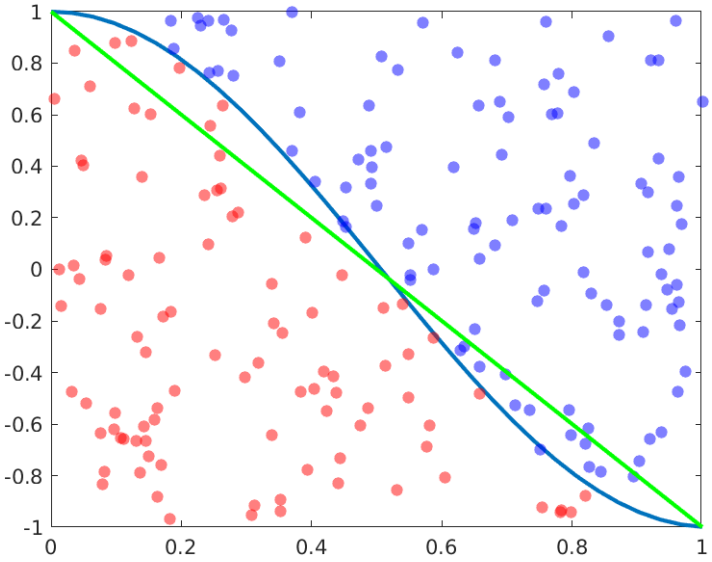
  $$F_\beta = \frac{\left(1 + \beta^2\right) \cdot precision \cdot recall}{\beta^2 precision + recall}.$$

7 Ensemble methods: increasing accuracy

An **ensemble method** for classification is a composite model, made up of a combination of classifiers. The idea is that given an unseen record, it can be classified using several models, and then make a consensus between all of them to decide the final decision of the class of the record. Usually, a voting scheme is used, in which the most voted class is the chosen one for classification. This approach usually improves the accuracy of each component.
Let's analyze why this works! There are three primary primary components to the error of a classifier:

1. **Bias**: every classifier has its own assumptions about the nature of the decision boundary between classes. When a classifier has high bias, it will make consistently incorrect predictions over records that lie near the incorrectly-modeled decision boundary.

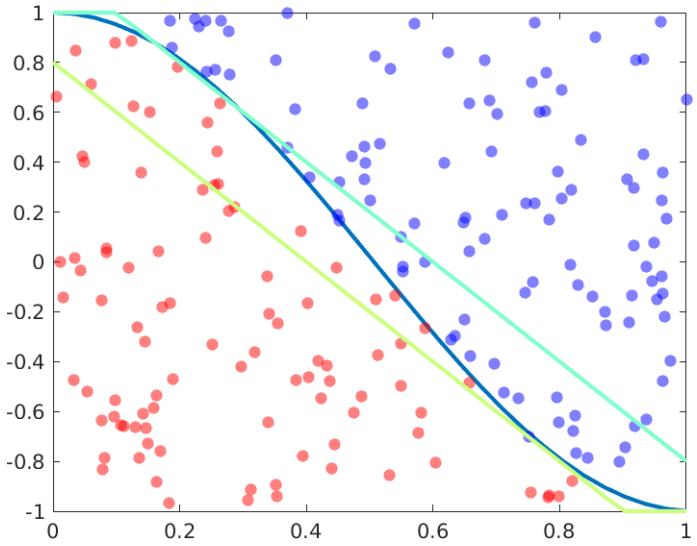
   Example 7.1. Bias is shown below.



The real model has been generated using the blue line. The classification model has the assumption that the data can be classified using a straight line. As we can see, points near to the boundary would be missclassified.

2. **Variance**: random variations in the choices of the traiing data will lead to different models. This is closely related to overfitting. When a classifier has an overfitting tendency, it will make inconsistent predictions for the same test instance oevr different training data sets.

   Example 7.2. Variance is shown below.



   In this case, the yellow-ish model happen to have used few red points near the boundary, so it became a perfect blue classifier, but a bad red classifier. The opposite happened to the blue-ish model. Note that this variations are only due to random selection of the training points.

3. **Noise**: it is the intrinsic errors in the target class labeling. As this is intrinsic, there is not much one can do. Therefore, we will focus in the two latter sources of error.

In addition, bias and variance are often in a trade-off relationship: improving bias worsens variance, and vice-versa. Generally speaking, simplified assumptions about the decision boundary lead to greater bias but lower variance, while complex assumptions reduce bias but are harder to robustly estimate with limited data.

Ensemble analysis can often be used to reduce both the bias and variance of the classification process, because a combination of different simple models with high bias and little variance will reduce the bias as the assumptions will be combined to model more complex scenarios. On the other hand, a combination of complex models with low bias and high variance, will reduce the variance because decisions made by multiple models tend to be more consistent than those made by individual models.

Looking at the accuracy

Now, let's look at the accuracy of the ensemble model in comparison to its component models in the case of binary classification (for ease). Let's say the ensemble model is composed of N models, each of them with an accuracy acc_i. Then, for the ensemble model to classify a new record as correct, at least half of the models need to classify the tuple correctly. Let's define the random variable X as '*number of classifiers that classify the tuple as correct*', then

$$P\left(correct\right) = P\left(X \geq \frac{N}{2}\right) = 1 - P\left(X < \frac{N}{2}\right),$$

and we can decompose

$$P\left(X < \frac{N}{2}\right) = P\left(X = 1\right) + P\left(X = 2\right) + ... + P\left(X = \frac{N}{2} - 1\right).$$

Here, assuming independence of the different models, it is

$$P\left(X = 1\right) = \sum_i acc_i \times \prod_{j \neq i} \left(1 - acc_j\right),$$

$$P\left(X = 2\right) = \sum_i \sum_{j \neq i} acc_i \times acc_j \times \prod_{k \neq i,j} \left(1 - acc_k\right),$$

and so on. This yields to a complex formula, but when the case is the simplest in which $A = acc_i = acc_j, \forall i, j$, we are in a binomial distribution, having

$$P\left(X = k\right) = \binom{N}{k} A^k \left(1 - A\right)^{N-k}.$$

Example 7.3. Suppose an ensemble model with $N = 3$ and equal accuracy for the three models, $A > 0$. Then, the accuracy for the ensemble model is

$$Acc = P\left(correct\right) = 1 - P\left(X = 1\right) = 1 - 3A\left(1 - A\right)^2,$$

which can be compared with the individual accuracies:

$$Acc - A = 1 - 3A\left(1 - A\right)^2 - A = -3A\left(1 - A\right)^2 + 1 - A = \left(1 - A\right)\left[1 - 3A\left(1 - A\right)\right].$$

At this point, the accuracy will be increased whenever

$$1 - 3A\left(1 - A\right) > 0 \iff 3A^2 - 3A + 1 > 0.$$

The discriminant of this polynomial is

$$b^2 - 4ac = 9 - 12 = -3 < 0,$$

so all its solutions are complex and it does not cut the X axis. As the leading coefficient is positive, this polynomial is always positive and we conclude that the 3-ensemble method always improves the accuracy of the individual models.

Example 7.4. Suppose an ensemble model with $N = 5$ and equal accuracy for the three models, $A > 0$. Then, the accuracy for the ensemble model is

$$\begin{aligned} Acc = 1 - P\left(X = 1\right) - P\left(X = 2\right) &= 1 - 5 \cdot A\left(1 - A\right)^4 - \frac{5!}{2!3!} A^2 \left(1 - A\right)^3 \\ &= 1 - 5A\left(1 - A\right)^4 - 10A^2\left(1 - A\right)^3, \end{aligned}$$

and when compared to the individual values we obtain

$$\begin{aligned} Acc - A &= 1 - 5A\left(1-A\right)^4 - 10A^2\left(1-A\right)^3 - A \\ &= \left(1-A\right)\left[1 - 5A\left(1-A\right)^3 - 10A^2\left(1-A\right)^2\right]. \end{aligned}$$

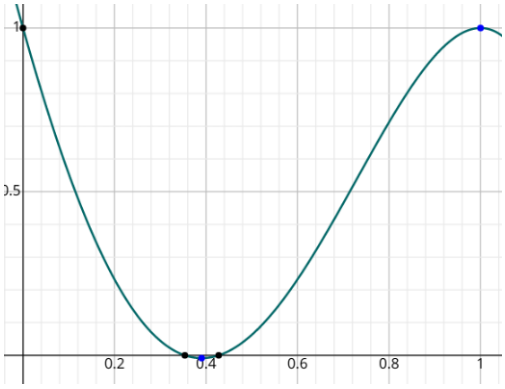
This is an increased accuracy whenever

$$1 - 5A\left(1-A\right)^3 - 10A^2\left(1-A\right)^2 > 0,$$

which corresponds to the polynomial

$$-5A^4 + 5A^3 + 5A^2 - 5A + 1 > 0.$$

The graph of this polynomial is



So for $A \in (0,1)$ it is positive everywhere except the interval $(0.354, 0.423)$, in which it is negative. This means that the accuracy is improved in all cases outside this interval.
For example, if $A = 0.7$, then $Acc = 0.839$, which is indeed an improvement.

There are several ways to make ensemble methods:

1. **Bagging**: the data is bootstrapped N times, collecting a training data set of approximately the same size of the original data set for each model. Then, each model is trained with a different bootstrap.

   This approach reduces the variance, because the differences owed to the sampling are reduced by performing this sampling N times. However, bagging does not improve the bias, because all the models used have the same assumptions.

2. **Boosting**: a weight is associated with each training instance and the N classifiers are trained with the use of these weights. When the classifier M_i has finished its training, those records in which the model missclassify are increased in weight, so the next model M_{i+1} will be trained paying more attention to those records.

   With this approach, the overall bias is reduced because each model focuses on those places where the past models tend to fail.

   (a) **Adaboost**: a particular algorithm approach based on the boosting idea. Given a dataset, $\mathcal{D}$, of k records with label y_i, $(X_1, y_1), ..., (X_k, y_k)$:

   i. Set initial weights to $w_i = \frac{1}{k}$, $\forall i = 1, ..., k$.
   ii. j = 1
   iii. While j<N
   A. Bootstrap $\mathcal{D}$ to get a training set $\mathcal{T}_j$, select tuples with probability w_i.
   B. Train model M_j with $\mathcal{T}_j$.
   C. Compute the error rate of M_j using $\mathcal{D}$.
   D. Update the weigths
   - If a tuple is misclassified: increase its weight
   - If it is correctly classified: decrease its weight

E. j = j+1

The error rate with weights is computed as

$$ER(M_j) = \sum_{i=1}^{d} w_i \cdot error(M_j, X_i),$$

where $error(M, X) = \begin{cases} 0 & if\ M\ classifies\ X\ correctly \\ 1 & if\ M\ misclassifies\ X \end{cases}$.

When the voting is performed, the votes are also weighted, with

$$w(M_i) = \log \frac{1 - ER(M_i)}{ER(M_i)}.$$

3. **Random Forest**: bagging does not work very well combined with decision trees, because the ID3 algorithm tends to generate similar/correlated trees. The idea here is to add randomness to the tree induction algorithm itself, as follows:

   (a) Before each split, L attributes are randomly selected out of the available K attributes.

   (b) The split attribute is selected from this group of L attributes.

   When L is selected to be much smalles than K, the trees in the forest are highly independent, so the method is able to improve the accuracy of the individual trees. Note nonetheless that in this case the individual trees will perform worse than normally trained trees, because they have been worsened on purpose.

Part III
Model validation and data preparation

8 Data preparation

The **data preparation phase** is a multistage process that comprises several individual steps, some or all of which may be used in a given application. These steps are:

1. **Feature extraction and portability**: a **feature** is characteristic of the data or derived from the data. For example, if we have a sensor measuring humidity, the level of humidity will be a feature directly present in the data; the difference between the humidity level at each measure and the average humidity level is a derived feature.

   Features with good semantic interpretability are more desirable because this makes things easier for the analyst to understand results. So, the process of selecting which features to take into account for further analysis is called **feature extraction**.

   Data type portability refers to the process of transforming data into different formats. This could have several reasons behind: we could do this because we have several sources of data which we want to unify or because we use an internal datatype that will not be compatible with what the training algorithms expect.

2. **Data cleaning**: missing, erroneus and inconsistent entries are treated. We can either remove them or estimate them via the process of **imputation**.

3. **Data reduction, selection and transformation**: the size of the data is reduced through data subset selection, feature subset selection, or data transformation. This helps in two ways:

   (a) The algorithms perform more efficiently in smaller datasets.

   (b) The removal of irrelevant features or records improves the quality of the data mining process.

8.1 Feature extraction

Example 8.1. Image feature extraction
Image data are represented as pixel. Nonetheless, we know that pixels are related between each others and that combinations of pixels carry information about what we are seeing in the image. This is not straighforward for a computer to understand, as the computer only 'sees' a matrix of triplets.
At a higher level, **color histograms** can be used to represent the features in different segments of an image.
Also, **visual words** are used to extract features from images. A visual word is a semantically rich representation of parts of an image.

Example 8.2. Document feature extraction
Document data is often available in raw and unstructured form, and the data may contain rich linguistic relations between different entities.
One approach is to remove stop words, stem the data, and use bag-of-words representation.
Other methods use entity extraction to determine linguistic relationships.
Named-entity recognition is an important subtask of information extraction. It consists in locating and classifying atomic elements in text into predefined expressions of names of persons, organizations,...

8.2 Data Type Portability

- **Numerical to categorizal data: discretizacion**

  The process of discretization divides the ranges of the numeric attribute into m ranges. Then, the attribute is assumed to contain m different categorical labeled values from 1 to m.

  Variations within a range are not ditinguishable after discretization (some information is lost).

  One challenge is that data may be nonuniformly distributed across the different intervals. Thus, there are several ways to perform the division:

- **Equi-width ranges**: the interval is divided into m subintervals of equal length.
- **Equi-log ranges**: the interval is divided in such a way that the log-length is constant. If we want to divide the interval $[a, b]$ into m equi-log ranges $\{[a_i, b_i]\}_{i=1}^{m}$, we have to ensure that

$$\log(b_i) - \log(a_i) = \log(b_j) - \log(a_j), \ \forall i, j = 1, ..., m$$

- **Equi-depth ranges**: the ranges are selected so that each range has an equal number of records.

- **Categorical to numeric data: binarization**

  Suppose a categorical attribute with m different values. Then, we can binarize it by creating m attributes, and the record will have all of them set to 0, except the one corresponding to the value that it has, which will be set to 1. For example:

Name	Role
Pedro	CEO

$\Longrightarrow$

Name	Role:CEO	Role:Employee	Role:Director
Pedro	1	0	0

- **Text to numeric data: Latent Semantic Analysis (LSA)**

  LSA transforms the text collection to a nonsparse representation with lower dimensionality. After transformation, each document $X = (x_1, ..., x_d)$ needs to be scaled to

$$\frac{1}{\sqrt{\sum_{i=1}^{d} {x_i}^2}} (x_1, ..., x_d).$$

  This is necessary to ensure that documents of varying length are treated in uniform way.

- **Time series to Discrete Sequence Data: Symbolic Aggregate Approximation (SAX)**

  Two steps:

  - **Window-based averaging:** the series is divided into windows of length w, and the average time-series value over each window is computed.
  - **Value-based discretization:** the averaged time-series values are discretized into a smaller number of equi-depth intervals. **Idea**: ensure that each symbol has an approximately equal frequency in the time-series.

- **Time series to Numeric Data:**

  - **Discrete Wavelet Transform (DWT)**: converts the time series data to multidimensional data, as a set of coefficients that represent averaged differences between different portions of the series.
  - **Discrete Fourier Transform (DFT)**: similar, using Fourier series' theory.

- **Discrete Sequence to Numeric Data:**

  Two steps:

  - Convert the discrete sequence to a set of binary time series, with as many time series as the number of values the discrete sequence can take.
  - H each of these time series into a multidimensional vector using the DWT. This combines the features from the different series, creating a single multidimensional record.

- **Spatial to Numeric Data:**

  The approach is the same as the one used for time-series data, but now there are two contextual attributes instead of one, so the DWT has to be modified to be two-dimensional.

8.3 Data Cleaning

Data in the real world is

- **incomplete**: there are values for some attributes that are missing. This can happen because some measures were not always taken of because of human/computer errors.
- **noisy**: there are errors. This can happen because the instruments used to collect data are not working properly, because errors in the data transmission occur or because of human/computer errors.
- **inconsistent**: there are discrepancies between different attributes that are related. This can happen when data from different sources needs to be combined or when some calculated values are not updated after changing their source values.
- **duplicate**: there are duplicate values.

8.3.1 Handling Missing and Inconsistent entries

We can:

- Delete the entire record: this is a safe option, because we don't introduce bias, but it usually not practical, because we might delete too many entries.
- Impute/estimate the missing values: we use the rest of the data to estimate the values that are missing or choose some constant based on some assumption. The problem with this approach is that one way or another we are introducing bias in the data.
- Change the mining algorithm: there are some algorithms that are developed to deal with missing values.

8.3.2 Handling Noisy entries

We can:

- Perform kernel smoothing: for numerical data, we can use a kernel function to smooth the data values.
  - kNN smoother: replaces each value with the average of itself and its k-nearest neighbors.
  - kernel average smoother: replace a value with the weighted average of itself and its neighbors in a fixed size window.
- Binning: it is also possible to sort the data and partition it into equally sized bins. Then, the data can be smoothed by the bin mean, median or boundary values.
- Regression: smooth by fitting the data to a regression function.
- Change the mining algorithm: there are algorithms designed to tolerate noise.

8.4 Exploratory analysis

Exploratory analysis is the task to understand the data, from the meaning of the features, to their range of values or even their statistical distributions. There are many actions we can do to explore the data, such as counting nulls, searching for repetition, compute some statistics as the maximum, the minimum, the mean,... of the data, and many more.

8.4.1 Central tendency measures

Central tendency measures are a 1 number summary that can be helpful:

- Mean:
$$\overline{X} = \frac{\sum X_i}{N}.$$

- Weighted mean:
$$\tilde{X} = \frac{\sum w_i X_i}{\sum w_i}.$$
- Trimmed mean: a mean calculated disregarding extreme values.
- Median: the middle value of the data.
- Mode: value that occurs most frequently in the data.

8.4.2 Symmetric and Skewed data

Using the mean, median and mode, we can understand in a soft way the distribution of the data:

- If the three values are very similar, then the distribution is very symmetric, having this values in the center.
- If the order is
$$Mode < Median < Mean,$$
then the distribution is skewed to the left.
- If the order is
$$Mean < Median < Mode,$$
then the distribution is skewed to the right.

8.4.3 Measuring the dispersion

- Quartiles: the Q1 (25th percentile) and Q3 (75th) percentile.
- Inter-quartile range:
$$IQR = Q_3 - Q_1$$
- Five number summary: minimum, Q1, median, Q3, maximum.
- Boxplot: the median is marked and there is a box around it which ends in the queartiles. Outliers are plotted individually.
- Outlier: a value that lies outside the range $1.5 \times IQR$.

8.4.4 Comparing with the normal distribution

We can compute the mean μ and the standard deviation σ and check if the data behaves as a normal distribution:

- In $(\mu - \sigma, \mu + \sigma)$ there is about 68% of the data.
- In $(\mu - 2\sigma, \mu + 2\sigma)$ there is about 96% of the data.
- In $(\mu - 3\sigma, \mu + 3\sigma)$ there is about 99.7% of the data.

We can also perform the Kolmogorov-Smirnov test or the Shapiro-Wilk test, which are statistical test that try to assess is a distribution is normal.

8.5 Similarity and Distance

There are many data mining algorithms which uses the notions of similarity or distance between two points. Usually, the selection of the distance function is an important decision before using an algorithm, because it will ultimately influence the results and their implications.

Definition 8.1. The $\boldsymbol{L_p}$**-norm** is a distance function defined by

$$Dist\,(X,Y) = \left(\sum_{i=1}^{d} |X_i - Y_i|^p\right)^{\frac{1}{p}}.$$

For $p = 2$ it is the well-known Euclidean distance.
For $p = 1$ it is called the Manhattan distance, because it is like traversing a grid made of rectangles, similar to the streetmap of Manhattan.

Definition 8.2. The **generalized Minkowski distance** is defined by

$$Dist\,(X,Y) = \left(\sum_{i=1}^{d} a_i \cdot |X_i - Y_i|^p\right)^{\frac{1}{p}}.$$

Remark 8.1. As we can see Minkowski is a weighted L_p-norm. This is useful in context where some features are more important than others, so they can have a higher weight in the distance measures.

Remark 8.2. Generally, p is set to d, the number of dimensions of the data.

Remark 8.3. Multidimensional data has normally different scales for the different dimensions, resulting in features dominating others in distance computations. To solve this issue we can do **normalization** and **scaling.**

- Normalization is to replace each value X_i with

  $$Z_i^j = \frac{X_i^j - \mu_j}{\sigma_j},$$

  where μ_j is the mean of the attribute j and σ_j its standard deviation.

- Scaling maps the values to the range $[0, 1]$:

  $$Y_i^j = \frac{X_i^j - \min_j}{\max_j - \min_j}.$$

Definition 8.3. The **edit distance** is a distance defined over strings.
We have the operators:

- r: replace one character by another.
- i: insert one character.
- d: delete one character.

The edit distance between two strings s_1 and s_2 is the minimum amount of operations needed to convert s_1 to s_2.
The formula is

$$edit(s_1, s_2) = \begin{cases} |s_1| & if \ |s_2| = 0 \\ |s_2| & if \ |s_1| = 0 \\ edit(tail(s_1), tail(s_2)) & if \ s_1[0] = s_2[0] \\ 1 + \min \left\{ \begin{array}{c} edit(tail(s_1), s_2) \\ edit(s_1, tail(s_2)) \\ edit(tail(s_1), tail(s_2)) \end{array} \right\} & otherwise \end{cases}$$

where $tail(s)$ is the string s minus its first character.

Remark 8.4. If the edit distance is computed recursively, its complexity measures are:

- Time: $O(|s_1| \times |s_2|)$.
- Space: $O(|s_1| \times |s_2|)$.
- Backtrace (length of the longest backtracking path): $O(|s_1| + |s_2|)$.

Remark 8.5. If we do it with dynamic programming, with the algorithm in Algorithm 3, then its complexity measures are:

- Time: $O(|s_1| \times |s_2|)$.
- Space: $O(|s_1| \times |s_2|)$.
- Backtrace: 0.

Algorithm 3 DP_Edit(s1, s2)

```
for i=1 to |s1| do
        for j=1 to |s2| do
                m[i,j] = min{m[i-1,j-1] + if[(s1[i] = s2[j] then 0 else 1],
                                          m[i-1,j] + 1,
                                          m[i,j-1] +1
                                         }
return m[|s1|,|s2|]
```

Remark 8.6. There is a further improvement that can be made. If we perform the dynamic programming only using rows or columns, then we only need to store two of them: the current one and the previous one. The complexity measures in this case are:

- Time: $O(|s_1| \times |s_2|)$.
- Space: $O(\min(|s_1|, |s_2|))$.
- Backtrace: 0.

9 Model evaluation

Once we have trained a model, we want to assess how well it performs. This way, we can compare different models and discuss, quantitatively, which of them is preferrable for our purposes. Nonetheless, this task is not easy, and there are both methodological and quantification issues to take into account:

- **Methodological issues**: associated with dividing the labeled data appropriately into training and test segments for evaluation. The choice of methodology has a direct impact on the evaluation process, such as underestimation or overestimation of classifier accuracy. Several approaches are possible: **holdout**, **bootstrap** and **cross-validation**.
- **Quantification issues**: associated with providing a numercial measure for the quality of the method after a specific methodology for evaluation has been selected.

9.1 Holdout

The labeled data is randomly divided into two disjoint sets, corresponding to the training and test data. The training data is used to feed the training algorithm and produce a model, whose performance is assessed using the test data.
The approach can be repeated several times with multiple samples to provide a final estimate.

- **Problem**: classes that are overrepresented in the training data are underrepresented in the test data. This can have a significant impact when the original class distribution is imbalanced. The error estimates are pessimistic.

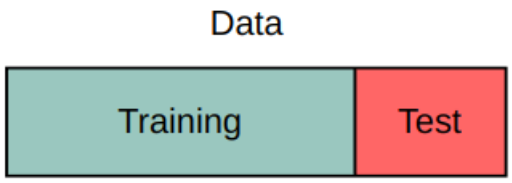


Figure 1: Holdout visualization.

9.2 Cross-Validation

The labeled data is divided into m disjoint subsets of equal size $\frac{n}{m}$. A typical choice of m is around 10. One of the m segments is used for testing, and the other $(m-1)$ segments are used for training. This approach is repeated by selecting each of the m different segments in the data as test set.
The average accuracy over the different test sets is then reported.
The overall accuracy of the cross.validation procedure tends to be a highly representative, but pessimistic estimate, of model accuracy.
When m is chosen to be $m = n$, $n-1$ examples are used for training, and one example is used for testing. This is called **leave-one-out cross-validation**. This approach is very expensive for large datasets.

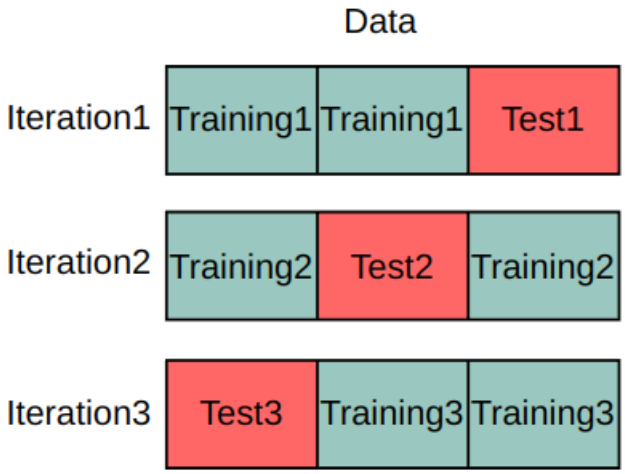


Figure 2: Cross-Validation visualization. $m = 3$.

Stratified cross-validation uses proportional representation of each class in the different folds and usually provides less pessimistic results.

9.3 Bootstrap

The labeled data is sampled uniformly with replacement, to create a training dataset, which can contain duplicates. The labeled data of size n is sampled n times with replacement.
The probability that a particular point is not included in a sample is

$$p_1 = 1 - \frac{1}{n}.$$

Therefore, the probability that the point is not included in n samples is

$$p_n = \left(1 - \frac{1}{n}\right)^n.$$

For large values of n, this approximates $\frac{1}{e}$. Thus, the fraction of the labeled data points included at elast once in the training dataset is $1 - \frac{1}{e}$.
The overall accuracy is computed using the original set of full labeled data as the test examples.
The estimate is highly optimistic of the true classifier accuracy because of the large overlap between the training and test examples.
A better strategy is the **leave-one-out bootstrap**, in which the accuracy of each labeled instance is computed using the classifier performance on only the subset of the bootstraped samples in which the instance is not part of.
This approach provides a pessimistic accuracy estimate, A_l, given by the mean value of the accuracy computed for each labeled instance.
The **0.632-bootstrap** improves the accuracy estimate with a compromise approach. The average training-data accuracy A_t over b bootstrapped samples is computed. This is a highly optimistic estimate. The overall accuracy is a weighted average of the leave-one-out accuracy and the training-data accuracy:

$$A = (0.632) \cdot A_l + (0.368) \cdot A_t.$$

Part IV
Clustering

Some applications require to divide the data into different groups, that share some characteristics. The problem many of these times is that we don't know which characteristics or at how much extend are useful to characterize the data. The general (unsupervised) approach to tackle this problem is **clustering**.

Definition 9.1. Clustering problem (Informal)
Given a set of data points, partition them into groups containing similar data points.

This definition is informal and general, but gives enough information to understand the problem, as well as enough freedom to tackle it from different perspectives.

10 Representative-Based Algorithms

These are the simplest of all clustering algorithms, as they directly use distances or similarities to cluster the data. They not capture hierarchical relationships and use a set of **representatives** to cluster the data. The main insight is that the discovery of good clusters equates to the discovery of good representatives.

Definition 10.1. Representative-Based general clustering problem
Given a data set $\mathcal{D}$ with n data points $X_1, ..., X_n$ in a d-dimensional space and a specified number of clusters, k, the goal of a representative-based algorithm is to determine k representatives $Y_1, ..., Y_n$ such that the objective function

$$O = \sum_{i=1}^{n} \left[\min_j \; Dist\left(X_i, Y_j\right) \right]$$

is minimized, i.e., the sum of the distances of the different data points to their closest representative needs to be miminized.

Remark 10.1. The representatives $Y_1, ..., Y_k$ and the optimal assigmnent of data points to representatives are unknown a priori, but they depend on each other in a circular way. This fact allows us to develop a iterative approach to solve the problem.

The **generic k-representative approach** is as in Listing 4.

Algorithm 4 Generic k-representative approach (Data D, int k, threshold eps) : Set of representatives Y and Clusters C

```
Initialize Y = {Y_1,...,Y_k} #Using heuristics
Initialize clusters C_1 = {},... C_k = {}
do
# Assign step
        for(X in D):
                assign X to Y_j such that Dist(X,Y_j) = min_i Dist(X,Y_i)
                C_j.add(X)

# Optimize step
        for all Clusters C_j:
                determine Y_j' such that
                        sum_{X_i in C_j} Dist(X_i,Y_j')
                is minimized
                Y_j = Y_j'

while O = sum_{i=1}^{n} [min_j Dist(X_i,Y_j)] > eps
return {C_1, Y_1},...,{C_k, Y_k}
```

Remark 10.2. The idea is to improve the objective function over multiple iterations. The increase is usually greater in early iterations, and decreases rapidly.

Remark 10.3. The main computational bottleneck is the assignment step, where distances need to be computed between all point to the representatives.

10.1 The k-Means algorithm

The $k-means$ algorithm is a representative clustering method in which the distance used is the squared Euclidean distance (or squared L_2-norm):

$$Dist\left(X_i, Y_j\right) = \|X_i - Y_j\|_2^2 .$$

Thus, the objective function minimizes the sum of square errors over the data points, this is called the SSE (Sum of Squared Errors).

Proposition 10.1. *The optimal representative Y_j for each of the optimize iterative steps is the mean of the data points in cluster C_j.*

Proof. In the current step, we have a fixed clustering asssignment from the last step, $C_1, ..., C_k$. The overall clustering objective function is

$$O\left(X, Y\right) = \sum_{j=1}^{k} \sum_{X_i \in C_j} \|X_i - Y_j\|_2^2 ,$$

so its gradient for each Y_j is

$$\frac{d}{dY_j} O\left(X, Y\right) = 2 \sum_{X_i \in C_j} \left(X_i - Y_j\right).$$

When imposing the gradient equals to 0 (for optimization purposes), we get

$$\sum_{X_i \in C_j} X_i - Y_j = 0,$$

or, equivalently,

$$\sum_{X_i \in C_j} X_i = |C_j| \, Y_j \implies Y_j = \frac{\sum_{X_i \in C_j} X_i}{|C_j|} = mean\left(C_j\right).$$

□

Remark 10.4. Note that the obtained representatives could be a point which is not a point in the data. This property sometimes is undesirable.

Remark 10.5. Note also that for the proof, we have suposed a numerical attribute. Computing the mean of different (for example) texts does not seem easy (think for example in the words *classification* and *regression*, they could be certainly clustered inside *data mining techniques*, but the latter is hardly the mean of the two former words).

Remark 10.6. Regarding time complexity:

- The assign step is $O(n \cdot k)$, as we have to compute for each point, k distances.
- The optimize step is $O(n)$, as we have to compute the k different means using all n points.
- Overall, then, it is $O(n \cdot k)$ per iteration. But usually few iterations are needed.

Remark 10.7. Disadvantages:

- All points are clustered and taken into account equally for the objective function. This makes $k-means$ sensitive to outliers, which introduces bias.
- In some situations we cannot compute the mean of the data points (as the previous example).
- We need to know k in advance.
- As we are minimizing the Euclidean distance, the algorithm is biased towards finding spherical clusters, because the sphere is the shape whose maximum distance to its center is constant:

$$\mathbb{S}^n(y,r) = \left\{ x \in \mathbb{R}^{n+1} : \sum_{i=1}^{n+1} |x_i - y_i| \leq r^2 \right\}.$$

Example 10.1. Apply the $k-means$ algorithm with $k=3$ to the dataset

$$D = \{A_1 = (2,10), A_2 = (2,5), A_3 = (8,4), A_4 = (5,8), A_5 = (7,5), A_6 = (6,4), A_7 = (1,2), A_8 = (4,9)\}$$

and using the seed $Y_1 = A_5,\ Y_2 = A_6$ and $Y_3 = A_8$.

k-Means algorithm example (Example 10.1)

Introduce the data

```
X1 = [2,10];
X2 = [2,5];
X3 = [8,4];
X4 = [5,8];
X5 = [7,5];
X6 = [6,4];
X7 = [1,2];
X8 = [4,9];

D = [X1;X2;X3;X4;X5;X6;X7;X8]
```

```
D = 8x2
     2    10
     2     5
     8     4
     5     8
```

```
     7     5
     6     4
     1     2
     4     9
```

```
Y = [X5; X6; X8]
```

```
Y = 3x2
     7     5
     6     4
     4     9
```

Compute the distance between all points

```
dist = zeros(3);

for j=1:3
    for i=1:8
        dist(i,j)=distance(D(i,:),Y(j,:),2);
    end
end
```

Select, for each point, the point that minimizes the distance

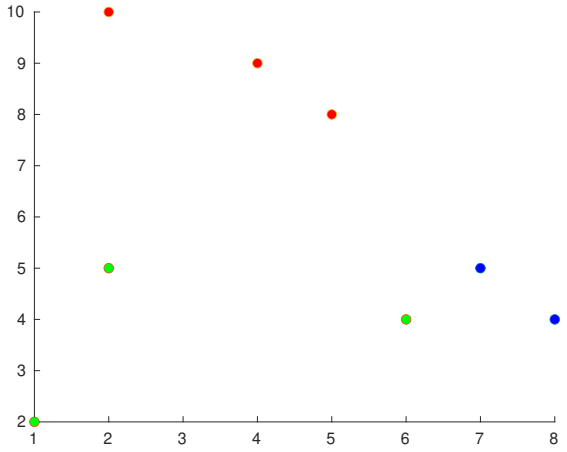
```
[v, idx] = min(dist, [], 2);
D = [D,idx];
```

Plot the clusters:

```
c1 = D(:,3) == 1
```

```
c1 = 8x1 logical array
   0
   0
   1
   0
   1
   0
   0
   0
```

```
c2 = D(:,3) == 2;
c3 = D(:,3) == 3;
scatter(D(c1,1),D(c1,2), 'MarkerFaceColor', 'b');
hold on
scatter(D(c2,1),D(c2,2), 'MarkerFaceColor', 'g');
scatter(D(c3,1),D(c3,2), 'MarkerFaceColor', 'r');
hold off
```



Compute the new `Y`:

```
for i=1:3
    ci = D(:,3) == i;
    x = mean(D(ci,1));
    y = mean(D(ci,2));
    Y(i,:)=[x,y];
end
Y
```

```
Y = 3x2
    7.5000    4.5000
    3.0000    3.6667
    3.6667    9.0000
```

Repeat the process:

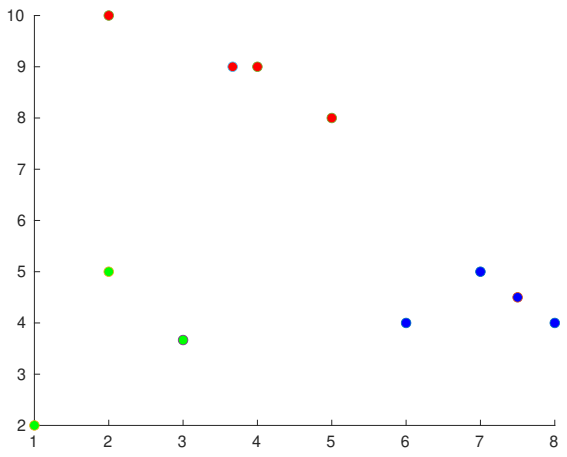
```
%Distances
for j=1:3
    for i=1:8
        dist(i,j)=distance(D(i,1:2),Y(j,:),2);
    end
end
dist
```

```
dist = 8x3
    7.7782    6.4118    1.9437
    5.5227    1.6667    4.3333
    0.7071    5.0111    6.6165
    4.3012    4.7726    1.6667
    0.7071    4.2164    5.2068
    1.5811    3.0185    5.5176
    6.9642    2.6034    7.4907
    5.7009    5.4263    0.3333
```

```
%Assign
[v, idx] = min(dist, [], 2);
D(:,3) = idx;
%Plot
c1 = D(:,3) == 1
```

```
c1 = 8x1 logical array
   0
   0
   1
   0
   1
   1
   0
   0
```

```
c2 = D(:,3) == 2;
c3 = D(:,3) == 3;
scatter(D(c1,1),D(c1,2), 'MarkerFaceColor', 'b');
hold on
scatter(Y(1,1),Y(1,2), 'MarkerFaceColor', 'b');
scatter(D(c2,1),D(c2,2), 'MarkerFaceColor', 'g');
scatter(Y(2,1),Y(2,2), 'MarkerFaceColor', 'g');
scatter(D(c3,1),D(c3,2), 'MarkerFaceColor', 'r');
scatter(Y(3,1),Y(3,2), 'MarkerFaceColor', 'r');
hold off
```




```
%Optimize
for i=1:3
    ci = D(:,3) == i;
    x = mean(D(ci,1));
    y = mean(D(ci,2));
    Y(i,:)=[x,y];
end
Y
```

```
Y = 3x2
    7.0000    4.3333
    1.5000    3.5000
    3.6667    9.0000
```

Again:

```
%Distances
for j=1:3
    for i=1:8
        dist(i,j)=distance(D(i,1:2),Y(j,:),2);
    end
end
dist
```

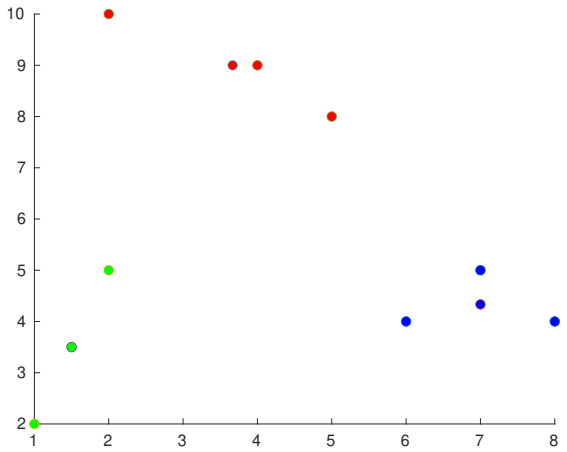
```
dist = 8x3
    7.5572    6.5192    1.9437
    5.0442    1.5811    4.3333
    1.0541    6.5192    6.6165
    4.1767    5.7009    1.6667
    0.6667    5.7009    5.2068
    1.0541    4.5277    5.5176
    6.4377    1.5811    7.4907
    5.5478    6.0415    0.3333
```

```
%Assign
[v, idx] = min(dist, [], 2);
D(:,3) = idx;
%Plot
c1 = D(:,3) == 1
```

```
c1 = 8x1 logical array
   0
   0
   1
   0
   1
   1
   0
   0
```

```
c2 = D(:,3) == 2;
c3 = D(:,3) == 3;
scatter(D(c1,1),D(c1,2), 'MarkerFaceColor', 'b');
hold on
scatter(Y(1,1),Y(1,2), 'MarkerFaceColor', 'b');
scatter(D(c2,1),D(c2,2), 'MarkerFaceColor', 'g');
scatter(Y(2,1),Y(2,2), 'MarkerFaceColor', 'g');
scatter(D(c3,1),D(c3,2), 'MarkerFaceColor', 'r');
scatter(Y(3,1),Y(3,2), 'MarkerFaceColor', 'r');
```

```
hold off
```




```
%Optimize
for i=1:3
    ci = D(:,3) == i;
    x = mean(D(ci,1));
    y = mean(D(ci,2));
    Y(i,:)=[x,y];
end
Y
```

```
Y = 3x2
    7.0000    4.3333
    1.5000    3.5000
    3.6667    9.0000
```

At this point we see how the clusters are the ones that we see naturally with our eyes. If we execute it again, we can see that the changes are very slight now:

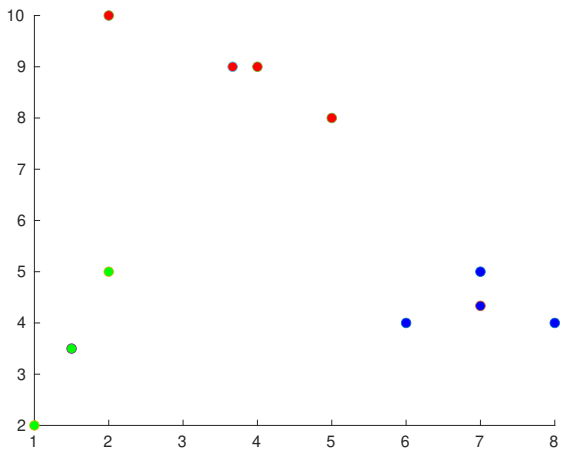
```
%Distances
for j=1:3
    for i=1:8
        dist(i,j)=distance(D(i,1:2),Y(j,:),2);
    end
end
dist
```

```
dist = 8x3
    7.5572    6.5192    1.9437
    5.0442    1.5811    4.3333
    1.0541    6.5192    6.6165
    4.1767    5.7009    1.6667
    0.6667    5.7009    5.2068
    1.0541    4.5277    5.5176
    6.4377    1.5811    7.4907
    5.5478    6.0415    0.3333
```

```
%Assign
[v, idx] = min(dist, [], 2);
D(:,3) = idx;
%Plot
c1 = D(:,3) == 1
```

```
c1 = 8x1 logical array
   0
   0
   1
   0
   1
   1
   0
   0
```

```
c2 = D(:,3) == 2;
c3 = D(:,3) == 3;
scatter(D(c1,1),D(c1,2), 'MarkerFaceColor', 'b');
hold on
scatter(Y(1,1),Y(1,2), 'MarkerFaceColor', 'b');
scatter(D(c2,1),D(c2,2), 'MarkerFaceColor', 'g');
scatter(Y(2,1),Y(2,2), 'MarkerFaceColor', 'g');
scatter(D(c3,1),D(c3,2), 'MarkerFaceColor', 'r');
scatter(Y(3,1),Y(3,2), 'MarkerFaceColor', 'r');
hold off
```




```
%Optimize
for i=1:3
    ci = D(:,3) == i;
    x = mean(D(ci,1));
    y = mean(D(ci,2));
    Y(i,:)=[x,y];
end
Y
```

```
Y = 3x2
    7.0000    4.3333
    1.5000    3.5000
    3.6667    9.0000
```

In fact, Y is not changing anymore!

Functions

```
function d = distance(X, Y, m)
    n = length(X);
    d = 0;
    for i=1:n
        d = d + (X(i)-Y(i))^m;
    end
    d = sqrt(d);
end
```

10.2 The k-Medians Algorithm

In this case the Manhattan distance (L_1-norm) is used:

$$Dist\left(X_i, Y_j\right) = \left\|X_i - Y_j\right\|_1 .$$

Proposition 10.2. *The optimal representative Y_j for each of the optimize iterative steps is the median of the data points along each dimension in cluster C_j.*

Proof. In this case, the objective function is

$$O\left(X, Y\right) = \sum_{j=1}^{k} \sum_{X_i \in C_j} \left\|X_i - Y_j\right\|_1 .$$

Now, the L_1-norm is obtained by summing the absolute value in each dimension. The problem is that this function is not differentiable. Nonetheless, it is differentiable almost everywhere. We can obtain the sub-gradient of O with respect to Y_j as

$$\frac{d}{dY_j} O\left(X, Y\right) = \sum_{X_i \in C_j} sign\left(X_i - Y_j\right) .$$

For this to equal 0, we need as many negative signs as positive signs: the median in each direction achieves exactly this, as it has as many values to its left as to its right. □

Example 10.2. Repeat the example using the k-Medians algorithm.

k-Medians algorithm example (Example 10.2)

Introduce the data

```
X1 = [2,10];
X2 = [2,5];
X3 = [8,4];
X4 = [5,8];
X5 = [7,5];
X6 = [6,4];
X7 = [1,2];
X8 = [4,9];

D = [X1;X2;X3;X4;X5;X6;X7;X8];
z = zeros([8,1]);
D = [D,z]
```

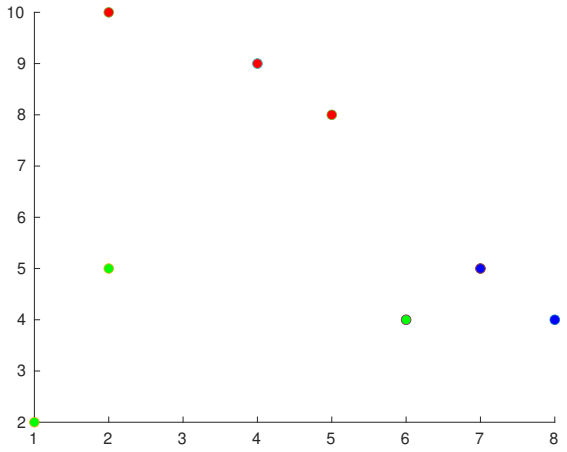
```
D = 8x3
     2    10     0
     2     5     0
     8     4     0
     5     8     0
     7     5     0
     6     4     0
     1     2     0
     4     9     0
```

```
Y = [X5; X6; X8]
```

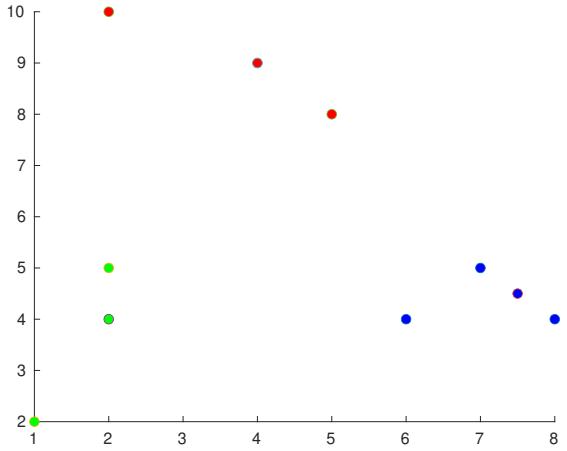
```
Y = 3x2
     7     5
     6     4
     4     9
```

Apply k-Medians

```
%First iteration
[D,Ynext] = kMediansIter(D,Y,3);
%Plot
c1 = D(:,3) == 1;
c2 = D(:,3) == 2;
c3 = D(:,3) == 3;
scatter(D(c1,1),D(c1,2), 'MarkerFaceColor', 'b');
hold on
scatter(Y(1,1),Y(1,2), 'MarkerFaceColor', 'b');
scatter(D(c2,1),D(c2,2), 'MarkerFaceColor', 'g');
scatter(Y(2,1),Y(2,2), 'MarkerFaceColor', 'g');
scatter(D(c3,1),D(c3,2), 'MarkerFaceColor', 'r');
scatter(Y(3,1),Y(3,2), 'MarkerFaceColor', 'r');
hold off
```

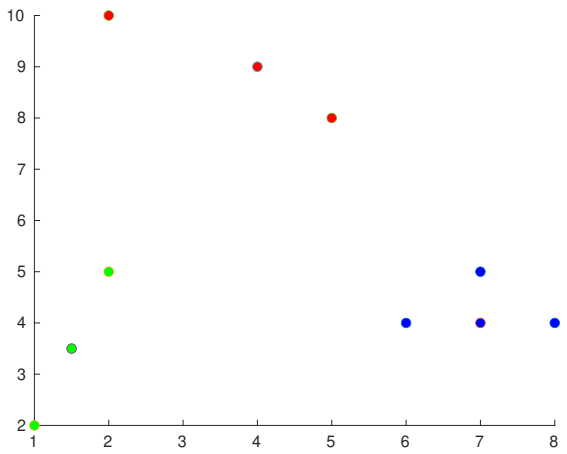



```
Y = Ynext;

%Second iteration
%First iteration
[D,Ynext] = kMediansIter(D,Y,3);
%Plot
c1 = D(:,3) == 1;
c2 = D(:,3) == 2;
c3 = D(:,3) == 3;
scatter(D(c1,1),D(c1,2), 'MarkerFaceColor', 'b');
hold on
scatter(Y(1,1),Y(1,2), 'MarkerFaceColor', 'b');
scatter(D(c2,1),D(c2,2), 'MarkerFaceColor', 'g');
scatter(Y(2,1),Y(2,2), 'MarkerFaceColor', 'g');
scatter(D(c3,1),D(c3,2), 'MarkerFaceColor', 'r');
scatter(Y(3,1),Y(3,2), 'MarkerFaceColor', 'r');
hold off
```




```
Y = Ynext;

%Third iteration
```

```
[D,Ynext] = kMediansIter(D,Y,3);
%Plot
c1 = D(:,3) == 1;
c2 = D(:,3) == 2;
c3 = D(:,3) == 3;
scatter(D(c1,1),D(c1,2), 'MarkerFaceColor', 'b');
hold on
scatter(Y(1,1),Y(1,2), 'MarkerFaceColor', 'b');
scatter(D(c2,1),D(c2,2), 'MarkerFaceColor', 'g');
scatter(Y(2,1),Y(2,2), 'MarkerFaceColor', 'g');
scatter(D(c3,1),D(c3,2), 'MarkerFaceColor', 'r');
scatter(Y(3,1),Y(3,2), 'MarkerFaceColor', 'r');
hold off
```




```
Y = Ynext;
```

```
function d = distance(X, Y, m)
    n = length(X);
    d = 0;
    for i=1:n
        d = d + abs(X(i)-Y(i))^m;
    end
    d = sqrt(d);
end

function [D2,Ynext] = kMediansIter(D,Y,k)
    dist = zeros(k);
    [n,m] = size(D);
    %Distances
    for j=1:k
        for i=1:n
            dist(i,j)=distance(D(i,1:2),Y(j,:),2);
        end
    end
    %Assign
    [v, idx] = min(dist, [], 2);
    D2 = D;
```

```
    D2(:,3) = idx;
    %Optimize
    Ynext = Y;
    for i=1:3
        ci = D2(:,3) == i;
        x = median(D(ci,1));
        y = median(D(ci,2));
        Ynext(i,:)=[x,y];
    end
end
```

10.3 The k-Medoids Algorithm

In this algorithm the representatives are always selected from the database $\mathcal{D}$, and this makes the structure of the algorithm different from the one we have seen before.
Reasons:

- This approach makes outlier handling easier than with k-means.
- It is sometimes difficult to compute the central representative of a complex data type (text or categorical data). The k-medoids algorithm can be defined in any datatype in which we are able to define a proper distance function.

The algorithm is as in Algorithm 5.

Algorithm 5 k-medoids(Data D, int k, threshold eps) : Set of representatives Y and Clusters C

```
Initialize Y = {Y_1,...,Y_k} #Using heuristics
Initialize clusters C_1 = {},... C_k = {}
do
# Assign step
        for(X in D):
                assign X to Y_j such that Dist(X,Y_j) = min_i Dist(X,Y_i)
                C_j.add(X)

# Optimize step
        Determine a pair X_i in D and Y_j in Y such that
                replacing Y_j with X_i leads to the
                greatest possible improvement in the objective function

        Perform the exchange between X_i and Y_j only if improvement is positive.

while O = sum_{i=1}^{n} [min_j Dist(X_i,Y_j)] > eps
                or no improvement in current iteration
return {C_1, Y_1},...,{C_k, Y_k}
```

Remark 10.8. In this algorithm we use a hill climbing strategy to obtain the best representatives.

Remark 10.9. We can try all possible changes or sample points from the database to try with. The latter approach is often more desirable for time issues.

10.4 Practical issues

- The initialization of the initial representative is not a trivial task. Normally they are selected randomly among the points of the datasets and most times the initialization step does not change the outcome of the algorithm: the representative based algorithm are very robust to this selection. Nonetheless, sometimes suboptimal clusters arise because of a bad choice of initial representatives.

- Outliers can make a detrimental impact on the algorithms. If one outlier is selected as initial representative it is possible to obtain a singleton cluster with no meaning for the application.
- The number of clusters, k, is difficult to determine using automated methods. As it is not known a priori, a common approach is to start with larger values for k than the one we think should be correct. Some natural clusters may split, but we can merge some of them as a postprocessing step.

11 Grid and Density based Algorithms

One of the major problems with distance-based algorithms is that the shape of the clusters is implicitly enforced by the distance function. Thus, it can be hard to detect natural cluster of arbitrary form.
Density-based algorithms are useful for this. The idea is to identify dense regions in the data, and use the positions of the different regions to determine the clusters.

11.1 Grid-based methods

The data is discretized into p intervals, typically equi-width. If the data has d dimensions, we will obtain p^d hyper-cubes. These are the building blocks for the clusters.
A **density threshold** τ is used to determine the dense hyper-cubes. In most real data-sets, an arbitrarily shaped cluster will result in multiple dense regions connected together by a side or a corner.
Two hyper-cubes are said to be **adjacently connected** if they share a side (sometimes corners are also considered).
Two hyper-cubes are said to be **density connected** if a path can be found from one to another containing only a sequence of adjacently connected grid regions.
The **goal** is to determine the density connected regions. Using a graph representation, the problem is equivalent to finding the connected components of the graph, being the hyper-cubes the nodes and an edge is defined between every pair of adjacent cubes.

Advantages
The number of clusters is not pre-defined, so we don't need to bother with the estimation of k.
Disadvantages
We have to define p and τ, which is not easy.

- If p (number of ranges) is too small, the data points from multiple clusters will be present in the same hyper-cube. We will obtain undesired merged clusters.
- If p is too large, there will be many empty grid cells, so we may split a natural cluster. It also will be computationally expensive.
- The choice of τ has similar consecuences.

Also, if the clusters present different densities, it is even more difficult to determine τ and p because each cluster is 'asking' for different values.

The generic algorithm is as follows:

Algorithm 6 GenericGrid(Data D, Ranges p, Density tau) : clusters C

```
Discretize each dimension into p ranges
Determine grid cells at density level tau
Create graph in which dense grids are connected if they are adjacent
Determine connected components of the graph
return points in each connected component as a cluster
```

11.2 DBSCAN

The idea behind DBSCAN is similar to the one we have seen, but density is considered at a pointwise level: The **density of a data point** is defined as the number of points that lie within a radius *eps* from it, i.e. their neighbourhood of radius τ. The densities are used to classify the points:

- **Core point**: its neighbourhood contains at least τ points.
- **Border point**: its neighbourhood contains less than τ points, but it contains one or more core points.
- **Noise points**: any other case.

And we define some relations between points:

- (p_i, p_j) are **directly density reachable** if p_i is a core point, and p_j is in the neighbourhood of p_i.
- (p_i, p_j) are **density reachable** is p_i is a core point, and there exists a chain of core points $p_{i+1}, ..., p_n$ where (p_k, p_{k+1}) are directly density reachable for $k = i, ..., n-1$ and p_j is in the neighbourhood of p_n.
- (p_i, p_j) are **density connected** if both p_i and p_j are density reachable from some point p_k

After the points have been classified, a connectivity graph is constructed as a maximal set of points that are all reachable from one another under any of these definitions.
Now, we identify the connected components of the graph, which are the clusters.
The detailed algorithm is as follows:

Algorithm 7 DBSCAN(Data D, Radius eps, Density tau) : clusters C

```
Clusters = {}
for each unvisited point P in D
        Neihbourhood = regionQuery(P, eps)
        if sizeof(Neihbourhood) < tau
                mark P as visited
        else
                C = next cluster
                expandCluster(P, Neighbourhood, C, eps, tau)
                if C not in Clusters
                        Clusters.add(C)
return Clusters

function expandCluster(P, Neighbourhood, C, eps, tau)
        mark P as visited
        C.add(P)
        for Q in Neighbourhood
                if Q not visited
                        mark Q as visited
                        Neighbourhood_Q = regionQuery(Q, eps)
                        if sizeof(Neighbourhood_Q >= tau)
                                Neighbourhood.addAll(Neighbourhood_Q)
                        if Q is not in any cluster
                                C.add(Q)
```

Advantages
This method is not very different from the graph method, and it can also discover clusters of any shape, without the need of knowing the number of clusters in advance.
Disadvantages
Again, determining the correct values for *eps* and τ is a complex task. Also, the existence of clusters with different densities makes it even harder.

Complexity

The major time complexity is finding the neighbours: $O\left(n^2\right)$.
In some special cases, an spatial index can reduce it to $O\left(n \cdot \log n\right)$.

Remark 11.1. Usually, grid based methods are more efficient because they partition the space, which makes the procedure less computationally expensive.

11.2.1 Progressive DBSCAN

The idea is using the same value for τ, apply DBSCAN in a progressive way, increasing the value of *eps*.

1. Start with a small *eps* to find dense clusters.
2. Iteratively relax the *eps* value to find less dense clusters.
3. After every iteration, the points that already belong to a cluster are removed from the dataset.

11.3 DENCLUE

The DENCLUE algorithm is based on kernel-density estimation, which can be used to create a smooth profile of the density distribution, by defining the density $f\left(X\right)$ at coordinate X as

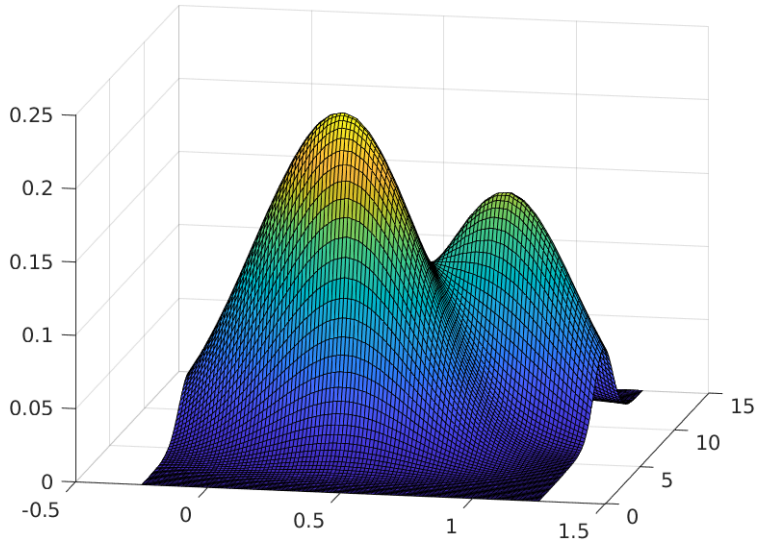
$$f\left(X\right) = \frac{1}{n}\sum_{i=1}^{n} K\left(X - X_i\right),$$

where K is the kernel function X_i are n different data points. A commonly used kernel function is the **Gaussian Kernel**:

$$K\left(X - X_i\right) = \left(\frac{1}{h\sqrt{2\pi}}\right)^d e^{-\frac{\|X - X_i\|^2}{2h^2}}.$$

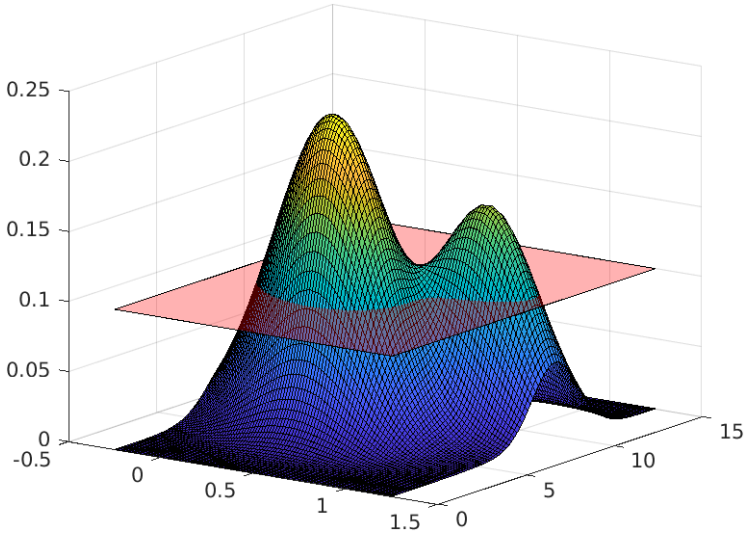
The effect of this operation is to replace each discrete data point with a smooth bump, and the density at each points is the sum of all these bumps.

Example 11.1. A visual example of a kernel smoothing:

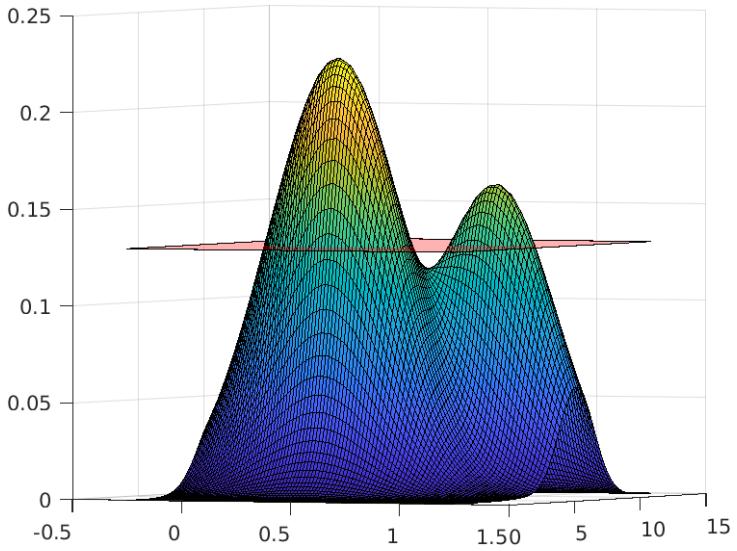


Once the density has been smoothed, the goal is to determine clusters by using a density threshold τ that intersects the density profile. Two examples showing how the choice of τ affects the result are shown in Example 11.2.

Example 11.2. In the previous example, if we select $\tau = 0.1$, we obtain the following:



In this case, only one cluster is obtained. In contrast, if we choose $\tau = 0.13$, two clusters are obtained:



12 Probabilistic Model-Based Algorithms

Until now, all models described are **hard clustering algorithm**, meaning each data point is assigned to a particular cluster. Probabilistic model-based algorithms are **soft algorithms**, in which each data point may have a nonzero assignment probability to more than one cluster.

12.1 Fuzzy sets and clusters

A **fuzzy cluster** is a fuzzy set $F_S : X \to [0, 1]$. For each data point $X_i \in X$, $F_S(X_i)$ represents the probability that X_i is in cluster S. $F_S(X_i)$ can be called **degree of membership** of object X_i to cluster S.
Formally, given a set of objects $X_1, ..., X_n$, a **fuzzy clustering** of k fuzzy clusters $C_1, ..., C_k$ can be represented using a **partition matrix**, $M = [w_{ij}]$, where

$$w_{ij} = F_{C_j}(X_i).$$

M should satisfy three conditions:

1. $w_{ij} \in [0, 1]$, $i = 1, ..., n, j = 1, ..., j$. From the definition of a fuzzy cluster.
2. $\sum_{j=1}^{k} w_{ij} = 1$, $i = 1, ..., n$. The sum of all probabilities is 1.

3. $0 < \sum_{i=1}^{n} w_{ij} < n,\ j = 1, ..., k$. There is no empty cluster.

12.2 Mixture model

The underlying assumption of a **mixture-based generative model** is to assume that the data was generated from a mixture of k distributions with probability distributions $G_1, ..., G_k$. Each of them represents a cluster and is called **mixture component**. The data points, X_i, are generated by this model as follows:

1. Select a mixture component with prior probability $\alpha_i = P(G_i)$. Say G_r is selected.
2. Generate a data point from G_r.

This generative model is denoted by $\mathcal{M}$. We don't know G_i nor α_i in advance. The G_i distributions are often assumed to be Gaussian[5], so we need to estimate the parameters of the distribution in such a way that the overall data has a maximum likelihood of being generated by the model.
Consider a set C of k probabilistic clusters $C_1, ..., C_k$ with probability density functions $f_1, ..., f_k$, respectively, and probabilities $p_1, ..., p_k$. The probability of an object X being generated by the cluster C_j is

$$P(X|C_j) = p_j \cdot f_j(X),$$

and the probability of X being generated by the set C is

$$P(X|C) = \sum_{j=1}^{k} p_j \cdot f_j(X).$$

As objects are assumed to be independently generated, for a data set $\mathcal{D} = \{X_1, ..., X_n\}$, the probability that $\mathcal{D}$ is generated by C is

$$P(\mathcal{D}|C) = \prod_{i=1}^{n} P(X_i|C) = \prod_{i=1}^{n} \sum_{j=1}^{k} p_j \cdot f_j(X_i).$$

Now, we want to estimate C from $\mathcal{D}$ trying to maximize $P(\mathcal{D}|C)$ is maximized.
If we use the assumption that the underlying distributions are Gaussian $\mathcal{G}(\mu_j, \sigma_j)$, then the probability density function of each cluster are centered at μ_j with standard deviation σ_j is:

$$P(X_i|\Theta_j) = \frac{1}{\sigma_j\sqrt{2\pi}} e^{-\frac{(X_i-\mu_j)^2}{2\sigma^2}}.$$

And if we assume all clusters have the same probability p_j, then

$$P(X_i|\Theta) = \sum_{j=1}^{k} \frac{1}{\sigma_j\sqrt{2\pi}} e^{-\frac{(X_i-\mu_j)^2}{2\sigma^2}}.$$

Thus, our objective is to maximize

$$P(\mathcal{D}|\Theta) = \prod_{i=1}^{n} \sum_{j=1}^{k} \frac{1}{\sigma_j\sqrt{2\pi}} e^{-\frac{(X_i-\mu_j)^2}{2\sigma^2}}.$$

This is achieved with the **expection-maximization (EM) algorithm**. The EM algorithm is a framework to approach maximum likelihood estimates of parameters in statistical models. It consists of two steps:

1. **E-step**: assigns objects to clusters according to the current fuzzy clustering or parameters of probabilistic clusters.
2. **M-step**: finds the new clustering or parameters that maximize the SSE or the expected likelihood.

Example 12.1. EM algorithm example.

[5]Note that any other distribution might be assumed.

EM Algorithm example (Example 12.1)

Start defining the data set:

```
D = [3,3;
    4,10;
    9,6;
    14,8;
    18,11;
    21,7];

centers = [3,3;
           4,10];

[p1,c1] = E_step(D, centers)
```

```
p1 = 2x6
    1.0000         0    0.4767    0.4160    0.4053    0.4671
         0    1.0000    0.5233    0.5840    0.5947    0.5329

c1 = 1x6
     1     2     2     2     2     2
```

```
centers_2 = M_step(D, p1)
```

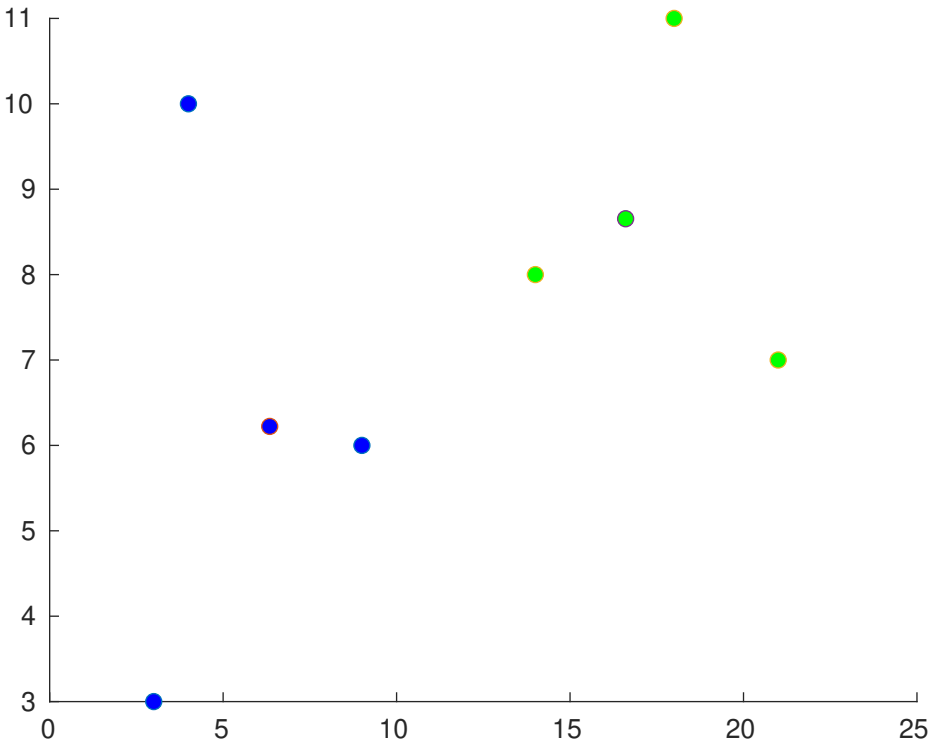
```
centers_2 = 2x2
    8.4178    5.0946
   10.4632    8.9897
```

```
[result, probs, cl] = em_algorithm(D, centers, 0.5)
```

```
result = 2x2
    6.3427    6.2224
   16.6020    8.6542

probs = 2x6
    0.9096    0.8905    0.9012    0.1043    0.0449    0.0930
    0.0904    0.1095    0.0988    0.8957    0.9551    0.9070

cl = 1x6
     1     1     1     2     2     2
```

```
D(:,3) = cl;
c1 = D(:,3) == 1;
c2 = D(:,3) == 2;

scatter(D(c1,1),D(c1,2), 'MarkerFaceColor', 'b');
hold on
scatter(result(1,1),result(1,2), 'MarkerFaceColor', 'b');
scatter(D(c2,1),D(c2,2), 'MarkerFaceColor', 'g');
scatter(result(2,1),result(2,2), 'MarkerFaceColor', 'g');
```

```
hold off
```




```
function d = distance(X, Y, m)
    n = length(X);
    d = 0;
    for i=1:n
        d = d + abs(X(i)-Y(i))^m;
    end
    d = sqrt(d);
end

function [probs,clusters] = E_step(data, centers)
    k = length(centers);
    n = length(data);
    distances = zeros([k,n]);
    for i = 1:k
        for j = 1:n
            distances(i,j) = distance(data(j,:),centers(i,:),2)^2;
        end
    end

    full_dists = sum(distances,1);
    probs = (full_dists - distances )./ full_dists;

    [c,idx] = min(distances,[],1);
    clusters = idx;
end
```

```
function new_centers = M_step(data,probs)
    sq_probs = probs.^2;
    sum_sq_probs = sum(sq_probs,2);
    new_centers = sq_probs * data;
    new_centers = new_centers ./ sum_sq_probs;
end

function [centers, probs, clusters] = em_algorithm(data, in_centers, eps)
    dif = inf;
    while dif > eps
        [probs,clusters] = E_step(data, in_centers);
        centers = M_step(data, probs);
        dif = sum(centers - in_centers);
        in_centers = centers;
    end
    [probs,clusters] = E_step(data, centers);
end
```

12.3 Evaluating fuzzy clusters

If $c_1, ..., c_k$ are the centers of the k clusters, we define the sum of squared error (SSE) for a point X_i as

$$SSE\left(X_i\right) = \sum_{j=1}^{k} w_{ij}^{p} \cdot dist\left(X_i, c_j\right)^2 .$$

For a cluster C_j, we have its SSE as

$$SSE\left(C_j\right) = \sum_{i=1}^{n} w_{ij}^{p} \cdot dist\left(X_i, c_j\right)^2 .$$

Finally, the SSE of the whole clustering is

$$SSE\left(\mathcal{C}\right) = \sum_{i=1}^{n} \sum_{j=1}^{k} w_{ij}^{p} \cdot dist\left(X_i, c_j\right)^2 .$$

12.4 Cluster quality measures

A good clustering methods will produce high quality clusters, i.e. clusters with the following characteristics:

- **High intra-class similarity**: cohesive within clusters. This means that the objects inside the clusters are similar to each other.
- **Low inter-class similarity**: distinctive between clusters. This means that the objects from different clusters are different.

The quality of the clustering method depends on the similarity measure used by the method, its implementation and its ability to discover the hidden patterns in the data.
Some examples of **quality measures** are:

- **Sum of square distance to centroids**: the squared distance between the representative of each cluster to every other point in the cluster, and then summed. This measure is suitable for representative-based methods, but it favors clusters that suit the underlying distance function used:

$$SSD\left(\mathcal{C}\right) = \sum_{j=1}^{k} \sum_{i=1}^{n_j} dist\left(X_i, c_j\right)^2 .$$

- **Intracluster to intercluster distance ratio**: we sample pairs of points from $\mathcal{D}$. Let P be the pairs (X_i, X_j) such that X_i and X_j are in the same cluster, and Q the pairs whose points are in different clusters. Then

$$IIDR = \frac{Intra}{Inter},$$

where

$$Intra = \sum_{(X_i,X_j)\in P} \frac{dist\,(X_i, X_j)}{|P|},$$

$$Inter = \sum_{(X_i,X_j)\in Q} \frac{dist\,(X_i, X_j)}{|Q|}.$$

- **Silhouette coefficient**: let D_{avg-in_i} denote the average distance between a point in the cluster i and the rest of the points in the same cluster. Let $D_{avg-out_{i,j}}$ denote the average distance between a point in the cluster i and every other point in the cluster j. Let $D_{min-out_i} = \min_j \left(D_{avg-out_{i,j}}\right)$. Then:

$$SC_i = \frac{D_{min-out_i} - D_{avg-in_i}}{\max\left\{D_{min-out_i}, D_{avg-in_i}\right\}}.$$

It follows that $SC_i \in (-1, 1)$, where large positive values indicate highly separated clusters (the distance to other clusters is high) and negative values are indicative of some level of mixing of data points from different clusters.

Part V
Frequent pattern and association rule mining

13 Frequent Itemset Mining

Association pattern mining is usually defined in the context of supermarket data coontaining sets of items bought by cutomers, which are referred to as **transactions**. The goal is to determine associations between groups of items bought by customers. The discovered sets of items are reffered to as **frequent itemsets**.
This frequent itemset can then be used to generate **association rules** of the form

$$X \implies Y,$$

where X and Y are sets of items. The meaning of this is that we discovered that when some customer buys X, it is likely that the same customer is going to/would like to by Y.
We have to be careful, nonetheless, because the raw frequency of a pattern is not the same as the statistical significance of the underlying correlations. This is why numerous models for frequent pattern mining have been proposed that are based on statistical significance.

Example 13.1. An intuitive example is that when someone buys bread, cheese and yogurt, it is probably the case that he will buy also milk and eggs:

$$\{Bread, Cheese, Yogurt\} \implies \{Milk, Eggs\}.$$

13.1 The model

The **problem of association pattern mining** is defined on *unordered set-wise data*.
The **database** $\mathcal{T}$ contains n transactions, $T_1, ..., T_n$.
Each **transaction** T_i is a subset of the set of all items $T_i \subset \mathcal{U}_{items} = \mathcal{U}$. The transactions can be represented as a multidimensional record of dimensionality $d = |\mathcal{U}|$, where the values are binary:

$$T_i\left(item\right) = \begin{cases} 1 & if\ item \in T_i \\ 0 & otherwise \end{cases}.$$

The universe of items is very large compared to the typical number of items in each transaction.

Definition 13.1. An **itemset** is a set of items, $I \subset \mathcal{U}$.
A ***k*-itemset** is an itemset with k items, $I \subset \mathcal{U} \wedge |I| = k$.
The **support** of an itemset I, $sup\left(I\right)$, is the fraction of the transactions in the database $\mathcal{T}$ that contain I as a subset.

Definition 13.2. The **frequent itemset mining problem** is defined as follows:
Given a set of transactions $\mathcal{T} = \{T_1, ..., T_n\}$, where each transaction T_i is a subset of items from $\mathcal{U}$, determine all itemsets I that occur as a subset of at least a predefined fraction **minsup** of the transactions in $\mathcal{T}$.
The predefined fraction minsup is called **minimum support**.
The unique identifier of a transaction is referred to as **transaction identifier** (tid).

Remark 13.1. The number of frequent itemset is generally very sensitive to the minimum suppor level:

- The use of a lower minimum support level, yields a larger number of frequent patterns.
- If the support level is too high, no frequent patterns will be found.

Therefore, an appropriate choice of the support level is crucial for discovering a set of frequent patterns with meaningful size.

Example 13.2. A very simple example:

Tid	Transaction	Binary Representation
1	{Shirt, Trousers}	110
2	{Shirt, Jacket}	101
3	{Shirt, Trousers, Jacket}	111

In this case, the possible itemsets and their respective support is:

Itemset	Support
{Shirt}	1
{Trousers}	$\frac{2}{3}$
{Jacket}	$\frac{2}{3}$
{Shirt, Trousers}	$\frac{2}{3}$
{Shirt, Jacket}	$\frac{2}{3}$
{Trousers, Jacket}	$\frac{1}{3}$
{Shirt, Trousers, Jacket}	$\frac{1}{3}$

If we select $minsup = \frac{1}{3}$, all possible itemsets would be selected.
If we select $minsup = 1$, only {Shirt} would be selected.

Now, let's think about how many possible itemsets there are: if an itemset if a subset of $\mathcal{U}$, then there are $2^{card(\mathcal{U})}$ possible itemsets. This means that computing all their supports as in the previous example would take an exponential amount of time to the cardinality of the number of items. Not only this, but the database needs also to be accessed for counting, comparing,... So it is easy to see how this problem becomes rapidly innaccessible. Then, it is compulsory to find better ways to perform this counting and comparing, or to be able to discard itemsets even before counting. For this, there are some very convenient properties of itemsets:

Property 1: Support Monotonicity Property
The support of every subset $J \subset I$ is at greater than or equal to the support of I:

$$sup\,(J) \geq sup\,(I)\,,\ \forall J \subset I.$$

This is because all subsets of the itemset I are also itemsets. As the itemset I appears in $sup\,(I)$ fraction of the total transactions, all its subsets also appears at least in the same transactions.

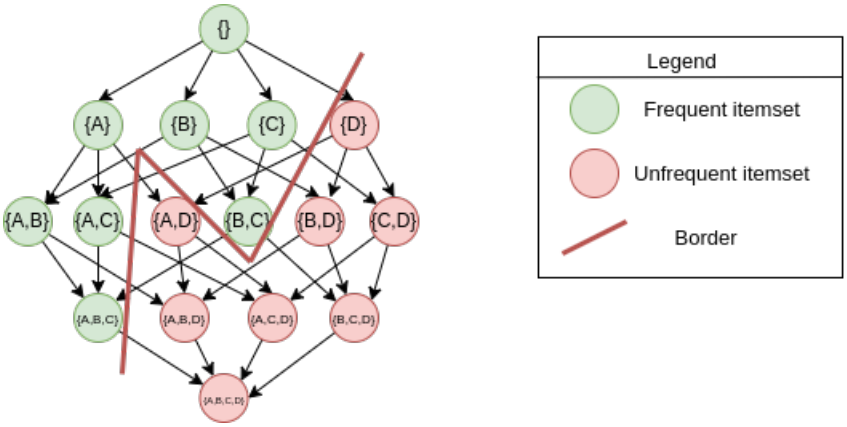
Property 2: Downward Closure Property
Every subset of a frequent itemset is also frequent.

This is a natural implication of the previous property, and it is very useful: when we discover a frequent itemset, we don't need to check its subsets because it is already assured that they are frequent, too.

Definition 13.3. A frequent itemset is **maximal** at a given minimum support level *minsup* if it is frequent, and no superset of it is also frequent.

The possible itemsets given a set of items can be conceptually arranged in the form of a **lattice of itemsets**, which contains one node for each of the $2^{|\mathcal{U}|}$ sets drawn from the universe of items. An edge exists between a pair of nodes if the corresponding sets differ by exactly one item. The lattice represents the search space of frequent patterns and it is separated into frequent and infrequent itemsets by a **border**.

Example 13.3. A lattice of itemset with 4 elements.



13.2 Association rule generation framework

Definition 13.4. Let X, Y be two itemsets. The **confidence of the rule**, $conf\,(X \implies Y)$, is the conditional probability of $X \cup Y$ ocurring in a transaction, given that the transaction contains X:

$$conf\,(X \implies Y) = Pr\,(X \cup Y | X) = \frac{sup\,(X \cup Y)}{sup\,(X)}.$$

X is called the **antecedent** of the rule and Y is the **consequent**.

Definition 13.5. Let X, Y be two itemsets. The rule $X \implies Y$ is said to be an **association rule at a minimum support of *minsup* and minimum confidence of *minconf*** if it satisfies:

1. The support of the itemset $X \cup Y$ is at least $minsup$.
2. The confidence of the rule $X \implies Y$ is at least $minconf$.

Here, the first criterion ensures that there are enough transactions to believe that the rule has statistical relevance. The second criterion ensures that the rule is strength enough in terms of conditional probabilities. The overall procedure for association rule generation uses two phases:

Phase 1) The frequent itemsets are generated at the minimum support of $minsup$.

Phase 2) The association rules are generated from the frequent itemsets at the minimum confidence level of $minconf$.

The first phase is more computationally intensive, so we are focusing on it from now on.

13.3 Frequent itemset mining algorithms

13.3.1 Brute Force Algorithms

For a universe of items $\mathcal{U}$, there are $2^{|\mathcal{U}|} - 1$ distinct subsets, excluding the empty set. A naïve idea would be to generate all these candidate itemsets and count their suppor against the transaction database $\mathcal{T}$.

Definition 13.6. A **candidate itemset** is an itemset that can be frequent, so it is needed to be checked.

Now, we can verified the candidates against the transaction database by support counting, checking whether a given itemset I is subset of each transaction $T_i \in \mathcal{T}$.
This approach is likely to be impractical when the universe of items is large.

A little tweak

The brute-force approach can be made faster by observing that no $(k+1)$-patterns are frequent if no k-patterns are frequent (this follows from the downward closure property). Thus, we can enumerate and count in increasing length for the patterns.
For sparse transaction databases, the value of l (the largest frequent itemset) is usually small compared to $|\mathcal{U}|$. At this point, one can terminate. This approach is orders of magnitude faster, but its computational complexity is still not satisfactory for large values of $\mathcal{U}$.

Ideas to apply

Better algorithms can be developed by using one or more of the following approaches:

1. Reduce the size of the explored search space by prunning cadidate itemsets using tricks.
2. Counting the support of each candidate more efficiently by prunning transactions that are know to be irrelevant.
3. Using compact data structures to represent either candidates of transaction databases that suppor efficient counting.

13.3.2 The Apriori Algorithm

The Apriori algorithm uses the downward closure property to prune candidates. If an itemset is infrequent, then all its supersets are also infrequent, so we don't need to count them.
The Apriori algorithm works as follows:

1. Count the support of the individual items to generate frequent 1-itemsets.
2. Combine the frequent 1-itemsets to generate candidate 2-itemsets.
3. Count the support of the candidate 2-itemsets to generate frequent 2-itemsets.
4. ...

In general:

1. Combine the frequent $(k-1)$-itemsets to generate candidate k-itemsets.
2. Prune candidate k-itemsets which has some subset which is not frequent.
3. Count the support of the candidate k-itemsets to generate frequent k-itemsets.
4. Repeat until there are no bigger frequent itemsets.

The algorithm is detailed in Algorithm 8.

Algorithm 8 Apriori(Transactions T, Minimum Support minsup)

```
k = 1
F1 = {Frequent 1-itemsets}

while Fk is not empty do
        Generate C(k+1) by joining itemset-pairs of Fk
        Prune itemsets from C(k+1) that violate the downward closure property
        Determine F(k+1) by support counting (C(k+1),T)
        k = k+1
end

return Union(F(i) for all i=1..k)
```

Remark 13.2. The downward closure property ensures that the candidate set generated does not miss any itemset that is frequent. This non-repetitive and exhaustive way of generating candidates can be interpreted in the context of a conceptual hierarchy of the patterns known as **enumeration tree**.

Remark 13.3. To do the pruning, we check generated elements against non-frequent itemsets already generated.

Remark 13.4. The support counting process is the most expensive part because it depends on the size of $\mathcal{T}$. The level-wise approach ensures that the algorithm is relatively efficient from a disk-access perspective: each set of candidates in C_k can be counted in a single pass over the data without the need for random disk accesses. Nonetheless, the counting procedure is still expensive.ç

Example 13.4. We are going to manually run an Apriori algorithm for a simple database using $minsup = 2$ (the minsup can be also indicated as a count, instead of as a frequency). The database is

Database	
TID	**Transaction**
1	A,B,E
2	B,D
3	B,C
4	A,B,D
5	A,C
6	B,C
7	A,C
8	A,B,C,E
9	A,B,C

Let's compute F_1 directly:

F1	
Itemset	**Count**
A	6
B	7
C	6
D	2
E	2

From this, we see that all 1-itemsets are frequent, so we are generating all possible 2-itemsets as C2, and counting them:

C2	
Itemsets	**Count**
A,B	4
A,C	4
A,D	1
A,E	2
B,C	4
B,D	2
B,E	2
C,D	0
C,E	1
D,E	0

$\Longrightarrow$

F2	
Itemsets	**Count**
A,B	4
A,C	4
A,E	2
B,C	4
B,D	2
B,E	2

We have coloured in red the infrequent itemset and generated F2. Now we can generate C3 by combining itemsets from F2. For this, we need to search for itemsets that share all values but one. For example: {A,B} and {A,C} are combinated to obtain {A,B,C}. After getting all possibilities, we find C3 and count:

C3	
Itemset	**Count**
A,B,C	2
A,B,D	
A,B,E	2
A,C,D	
A,C,E	
B,C,D	
B,C,E	
B,D,E	

$\Longrightarrow$

F3	
Itemset	**Count**
A,B,C	2
A,B,E	2

The purple coloured itemsets are pruned because they contain some of the infrequent itemsets of C2. The rest are still frequent, so we continue with the process. We combine the only two itemsets left to get {A,B,C,E} as C4. Note, nonetheless that {C,E} is an infrequent itemset of C2, so we will prune it and we are in fact done. The frequent itemsets with $minsup = 2$ are F1, F2 and F3.

Limits of Apriori

- Apriori makes use of the downward closure property (sometimes called apriori property), which is nice.
- The lattice is traversed in a general-to-specific way, it is possible to a bi-directional traversal.
- It is done in a breadth-first manner, it might be done in a depth-first way.
- The generate-and test strategy:
  - Generate is $O\left(\frac{|\mathcal{U}|}{k}\right)$ for k-candidates.
  - Test is the counting, in which we are traversing the whole database, $O(|\mathcal{T}|)$.

Apriori improved with tricks

In Algorithm 9 it is detailed an improved version of the Apriori algorithm.

Algorithm 9 Apriori_improved(Transactions T, Minimum support minsup)

```
        k = 1
        F1 = {Frequent 1-itemsets}

        while Fk is not empty do
                C(k+1) = generate(Lk)
                for tran in T do
                        C(tran) = subset(C(k+1),tran)
                        for cand in C(tran)
                                cand.count++
                        end
                end
                L(k+1)={cand in C(k+1) | cand.count >= minsup}
        end
        return Union(F(i) for i=1..k)

procedure generate(Lk)
        foreach itemset1 in Lk
                foreach itemset2 in Lk
                        if itemset1[1]=itemset2[1] and ... and itemset1[k-1] = itemset2[k-
                                c = itemset1 join itemset2
                                if has_infreq_subsets(c,Lk)
                                        continue
                                else
                                        add c to C(k+1)
                                end
                        end
                end
        end
        return C(k+1)

procedure has_infreq_subsets(c,Lk)
        foreach k-subset s of c
                if s not in Lk
                        return TRUE
                end
        end
        return FALSE
```

More Apriori Tricks

- Transaction reduction: a transaction that does not contain any frequent k-itemsets, cannot contain any frequent $(k+1)$-itemset either, so it can be removed to make counting faster.
- Partitioning: divide the database into a set of disjoint partitions. Find all frequent itemsets in every partition. A local frequent itemset may not be frequent for the whole database, but a frequent itemset must be frequent in at least one partition. The best about this approach is that it allows for parallel processing and the partition size can be selected for it to fit in memory.
- Sampling: perform Apriori on a random sample of the data. Rule accuracy is harmed, but efficiency is improved.

13.3.3 FP-Growth

Even though Apriori greatly improves the efficiency of the solution of the association pattern mining in comparison to the brute force approach, we have seen that it can still suffer from inefficiencies when the database

is big. Particularly, counting is very costly and when there are lots of items we will need to count many times. There is an improved solution for the problem: **Frequent Pattern Growth (FP-Growth)** which:

1. Transforms the database into a compressed data structur called **FP-Tree**, which retains frequent itemsets information.
2. Mines the FP-tree for frequent itemsets by:
   (a) Divide it into a set of conditional databases, the **conditional pattern bases**, each associated eith one frequent item.
   (b) Mines each of these patterns separately to generate association rules, without the need to re-count the original database.

Definition 13.7. A **FP-Tree** is a trie (prefix tree) data structure, which acts as a compressed representaiton of a conditional database.

- The path from the root to a leaf represents a repeated sub-transaction (frequent pattern) in the database.
- The path from the root to an internal node represents either a frequent pattern or a prefix.
- Each node is associated with a count, which is the number of transactions in the database containing the path to this node.
- The prefixes are sorted in the order from the most frequent to the least frequent to leverage the prefix-based comrpession.

The algorithm for FP-Growth is detailed in Algorithm 10.

Algorithm 10 FP-Growth(FP-Tree FPT, Minimum Support minsup, Current Suffix P)

```
if FPT is a single path
        determine all combinations C of nodes on the path, report Union(C,P) as frequent
else
        foreach item i in FPT do
                report Pi = Union(i,P) as frequent

                use pointers to extract conditional prefix paths from FPT containing i

                readjust counts of prefix paths
                remove i

                remove infrequent items from prefix paths

                reconstruct FPTi

                if FPTi not empty
                        FP–Growth(FPTi, minsup, Pi)
        end
end
```

Basically, what FP-Growth does is recursively find all frequent itemsets ending with a particular suffix by splitting the probelm into smaller subproblems.

Example 13.5. Let's use FP-Growth on the previous example database:

Database	
TID	Transaction
1	A,B,E
2	B,D
3	B,C
4	A,B,D
5	A,C
6	B,C
7	A,C
8	A,B,C,E
9	A,B,C

First, we need to construct the FP-Tree. For this purpose, we compute the frequent 1-itemsets and reorder them in the count order:

F1	
Itemset	**Count**
A	6
B	7
C	6
D	2
E	2

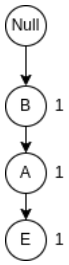
$\xrightarrow{\textit{In descending order}}$

F1	
Itemset	**Count**
B	7
A	6
C	6
D	2
E	2

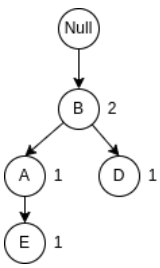
Now we reorder the transactions in the database following this same ordering:

Database	
TID	Transaction
1	B,A,E
2	B,D
3	B,C
4	B,A,D
5	A,C
6	B,C
7	A,C
8	B,A,C,E
9	B,A,C

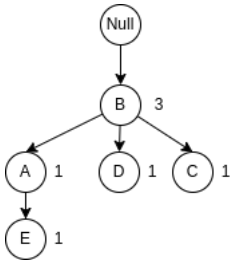
And now we construct the FP-Tree. We start with the root labeled as NULL, and then add the transactions one by one, reusing the prefixes when we can, and keeping a counter for each node. Every time we traverse a node, we increase the counter. The first transaction would be entered as:



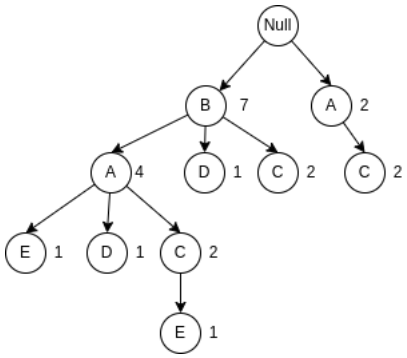
The second one:



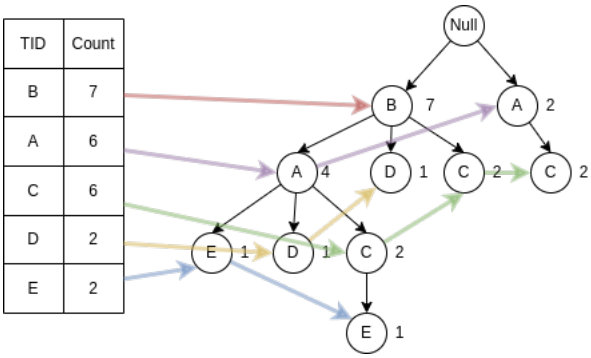
The third one:



And so on... Until all transactions are entered in the tree:

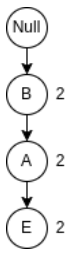


Now, we connect each item in the ordered F1 to one node in the database corresponding to the same item, and all nodes that are equal are connected, too. Like this:



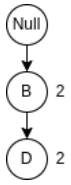
And now we proceed with the algorithm. We start with the least frequent item, E.

- For item E, the conditional pattern base contains two itemsets: $\{[\{B, A\} : 1], [\{B, A, C\} : 1]\}$. Thus, it follows that we have $\{[\{B, A\} : 2]\}$, so the conditional FP-Tree is



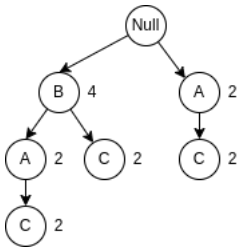
  Thus, the generated frequent patterns are: $\{[\{B, E\} : 2], [\{A, E\} : 2], [\{B, A, E\} : 2]\}$.

- For item D, the conditional pattern base also contains two itemsets: $\{[\{B, A\} : 1], [\{B\} : 1]\}$. So now we have $\{[\{B\} : 2]\}$ and the conditional FP-Tree is



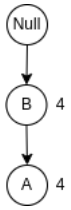
And the generated frequent patterns are $\{[\{B, D\} : 2]\}$.

- For item C, the conditional pattern base contains one itemset: $\{[\{B\} : 4]\}$. The conditional FP-Tree ends up being



And the generated frequent patterns are $\{[\{B, A, C\} : 2], [\{B, C\} : 2], [\{A, C\} : 2]\}$.

- For item A, the conditional pattern base contains three itemsets: $\{[\{B, A\} : 2], [\{B\} : 2], [\{A\} : 2]\}$. So now we have the three of them and no combination. The conditional FP-Tree ends up being



And the generated frequent patterns are $\{[\{B, A\} : 4]\}$.

And that's it!

Why FP-Growth outperforms Apriori?

- FP-Growth counts the database only once. This fact by itself is already a huge improvement, because Apriori needs to count one time per candidate.
- Also, Apriori needs indeed to create the candidates, which can also be expensive. FP-Growth, on the other hand, does not do a generation step, because the frequent itemsets are naturally obtained.
- In relation to the last point, Apriori creates candidates which end up being infrequent, while this does not happen with FP-Growth.
- Space complexity of FP-Growth is smaller because we don't need to store the candidates, only the FP-Tree, which is not much bigger than the number of items in $\mathcal{U}$. Also, the database is not needed after the FP-Tree is computed, so we have more memory to use for the tree itself.

13.4 Mining Association Rules

Once the frequent itemsets have been found, it is time to obtain the association rules, which are the goal we were aiming at since the beggining. To do this, a common approach is to follow:

1. For each frequent itemset I:
   (a) Generate subsets of I, $\emptyset \neq S \subset I$.
   (b) For every $\emptyset \neq S \subset I$:
      i. Output rule $S \implies I \setminus S$ if $conf(S \implies I \setminus S) \geq minconf$.

13.4.1 Evaluating association rules

In Definition 13.4 we saw the concept of confidence, but there are more measures to assess how 'good' a rule is, as the lift or the correlation analysis.

Definition 13.8. The **lift** is

$$lift(X \implies Y) = \frac{sup(X \cup Y)}{sup(X) \cdot sup(Y)}.$$

Remark 13.5. If $lift = 1$, then X and Y are independent.
If $lift > 1$, then X and Y have some dependency that is proportional to the lift value.
If $lift < 1$ then X and Y are contradicting, i.e., the existence of X discourages Y and vice versa.

Definition 13.9. The **correlation coefficient** is

$$\Phi = \frac{TT \cdot FF - TF \cdot FT}{\sqrt{\_T \cdot \_F \cdot T\_ \cdot F\_}},$$

where we are assuming a rule $A \implies B$ and:

- TT is how many times A is True and B is True
- FF is how many times A is False and B is False
- TF is how many times A is True and B is False
- FT is how many times A is False and B is True
- $T\_/F\_$ is how many times A is True/False
- $\_T/\_F$ is how many times B is True/False

Remark 13.6. In this case, if $\Phi = -1$ there is a perfect negative correlation.
If $\Phi = 1$ there is a perfect positive correlation.
If $\Phi = 0$ the two itemsets are statistically independent.

Remark 13.7. The definitions of TT, FT,... can be summarized as in the following table:

	B	¬B	Total
A	TT	TF	T_
¬A	FT	FF	F_
Total	_T	_F	

Example 13.6. Imagine we have the rule $\{Tea\} \implies \{Coffee\}$ with the following data:

	Coffee	¬Coffee	Total
Tea	150	50	200
¬Tea	650	150	800
Total	800	200	

In this case, the confidence is

$$conf(\{Tea\} \implies \{Coffee\}) = \frac{150}{200} = 0.75,$$

which is a high confidence... but $sup(Coffee) = 0.8$, which means that drinking coffee in fact decreases the probability of drinking coffee!
Now, the lift is

$$lift(\{Tea\} \implies \{Coffee\}) = \frac{0.15}{0.2 \cdot 0.8} = 0.9375 < 1,$$

which means that Tea and Coffee are negatively correlated. This insight is better than the one obtained by only looking at the value of the confidence.

Example 13.7. But let's now look at this example:

	p	¬p	Total		r	¬r	Total
q	880	50	930	s	20	50	70
¬q	50	20	70	¬s	50	880	930
Total	930	70		Total	70	930	

In this case

$$lift(\{p\} \implies \{q\}) = 1.02$$

and

$$lift(\{r\} \implies \{s\}) = 4.08,$$

but (p, q) appear together 88% of the time, while (r, s) appear together only 2% of the time. Confidence is a better indicator in this case. The problem here is that r, s appears in a small portion of records in the data.

13.4.2 How to choose a measure?

We have seen that we can obtain different conclusions by looking at different measures, so the choice of the measure is important because it will affect the results. A good choice must be based on a clear understanding of the measure and its properties, so we are aware of the flaws it entails and can leverage them for good.
We can also asses rules using interactive visualizations, subjective measures based on domain experience,...

14 Sequential pattern mining

In the last section, we were looking for patterns in a set-wise approach, but there exist also situations in which the order in which elements are encountered is important. For example, it is a well known fact that seeing something can make people wanting to buy it. This fact is used by supermarkets to strategically place the products in order for the customers to have as many temptations as possible. From the side of the supermarket, it is interesting to know which zones of the supermarket are related in such a way that customers tend to go from one to another in a particular order. Imagine we detect that it is usual that people that go to the desserts hall, usually go to the fruit hall afterwards. Then, the supermarket is interested in placing the fruits as far from the desserts as possible, to maximize the time spent by the customer at looking products that they did not want before, but might want now. It can be used also to detect sequences like:

- Shopping sequences: $computer \to printer \to ink$.
- Website sessions: $home \to education \to bachelor$.
- Tourist routes: $GrandPlace \to MannekenPis \to ComicsMuseum$.

With this objective in mind, **sequential pattern mining** was developed.

Definition 14.1. A **sequence** is an ordered list of elements. Each **element** is an unordered set of items.
A **subsequence** s of a sequence S is a sequence such that:

- If $e \in s$ then there exists $E \in S$ such that $e \subset E$.
- If $e_1 \to e_2 \to ... \to e_k$ is the full sequence s, then there exist k different elements $E_1, ..., E_k \in S$ such that $e_i \subset E_i, i = 1, ..., k$ and E_1 goes before E_2 (maybe not directly), E_2 goes before E_3,...

In other words, $s = [e_1, ..., e_k]$ is a subsequence of $S = [E_1, ..., E_m]$ if there are k elements $E_{s_1}, ..., E_{s_k} \in S$ such that $e_i \subset E_{s_i}, \forall i = 1, ..., k$ and $s_1 < s_2 < ... < s_k$.

Definition 14.2. A **sequence database** is a database of sequences.
The **support** of a sequence s is the fraction of sequences in the database $\mathcal{S} = \{S_1, ..., S_N\}$ that contain s as a subsequence.

Definition 14.3. The **sequential pattern mining problem** consists in, given a sequence database, find all subsequences whose support is at least the specified *minsup*.

This problem is obviously similar and related to that of frequent patterns mining, but this is more comples. A naive algorithm that generates all possible paterns and count them in the database would be highly inneficient, even more than in the set-wise case. There are several methods that work much better. First, let's see how to generate candidates.

14.1 Candidate generation

The candidate generation follows the steps:

1. **Base case**, $k = 2$: merging two frequent 1-sequences $[\{i_1\}]$ and $[\{i_2\}]$ will produce two candidate 2-sequences:
$$[\{i_1, i_2\}] \ and \ [\{i_1\}, \{i_2\}].$$
2. **General case**, $k > 2$: two frequent $(k-1)$-sequences w_1, w_2 are merged if the subsequence obtained by removing the first event in w_1 is the same as the subsequence obtained by removing the last event in w_2.
   (a) The resulting candidate after merging is given by the sequence w_1 extended with the last event of w_2:
      i. If the last two events in w_2 belong to the same element, then the las event in w_2 becomes part of the last element in w_1.
      ii. Otherwise, the last event in w_2 in w_2 becomes a separate element appended to the end of w_1.ç
      iii. If both w_1 and w_2 consist in only one set, we generate the two variants.

Example 14.1. If we merge $w_1 = [\{1\}\{2,3\}\{4\}]$ and $w_2 = [\{2,3\}\{4,5\}]$ we will get the sequence $w_3 = [\{1\}\{2,3\}\{4,5\}]$, because the last two events in w_2 belong to the same element.
If we merge $w_1 = [\{1\}\{2,3\}\{4\}]$ and $w_2 = [\{2,3\}\{4\}\{5\}]$ we will get the sequence $w_3 = [\{1\}\{2,3\}\{4\}\{5\}]$.

14.2 Generalized Sequential Pattern (GSP) Algorithm

1. Make the first pass over the database to obtain all the 1-element frequent sequences.
2. Repeat until no new frequent sequences are found:
   (a) **Candidate generation**: merge pairs of frequent subsequences found in the $(k-1)^{th}$ pass to generate candidate sequences that contain k items.
   (b) **Candidate pruning**: prune candidate k-sequences that contain infrequent $(k-1)$-subsequences.
   (c) **Support counting**: make a new pass over the sequence database to find the support for these candidate sequences.
   (d) **Candidate elimination**: eliminate candidate k-sequences whose actual support is less than *minsup*.

Example 14.2. Let's perform GSP with the following database with $minsup = 2$.

Database	
Id	**Sequence**
1	$[\{A,B\}\{C\}\{A\}]$
2	$[\{A,B\}\{B\}\{C\}]$
3	$[\{B\}\{C\}\{D\}]$
4	$[\{B\}\{A,B\}\{C\}]$

First, we compute C1 and F1:

C1	
Subsequence	**Count**
$[\{A\}]$	3
$[\{B\}]$	4
$[\{C\}]$	4
$[\{D\}]$	1

$\Longrightarrow$

F1	
Subsequence	**Count**
$[\{A\}]$	3
$[\{B\}]$	4
$[\{C\}]$	4

Now, we generate C2 by combining these subsequences. For example, merging $[\{A\}]$ and $[\{B\}]$ gives $[\{A\}\{B\}]$ and $[\{A, B\}]$, but we also need to combine in the other possible order, obtaining $[\{B\}\{A\}]$ in addition to these. Thus, we obtain:

C2	
Subsequence	**Count**
$[\{A\}\{A\}]$	1
$[\{A\}\{B\}]$	1
$[\{A, B\}]$	3
$[\{B\}\{A\}]$	2
$[\{B\}\{B\}]$	2
$[\{A\}\{C\}]$	3
$[\{A, C\}]$	0
$[\{C\}\{A\}]$	1
$[\{C\}\{C\}]$	0
$[\{B\}\{C\}]$	4
$[\{B, C\}]$	0
$[\{C\}\{B\}]$	0

$\Longrightarrow$

F2	
Subsequence	**Count**
$[\{A, B\}]$	3
$[\{B\}\{A\}]$	2
$[\{B\}\{B\}]$	2
$[\{A\}\{C\}]$	3
$[\{B\}\{C\}]$	4

Now, from F2 we can combine $[\{A, B\}]$ with $[\{B\}\{A\}]$, $[\{B\}\{B\}]$ and $[\{B\}\{C\}]$, $[\{B\}\{A\}]$ with $[\{A, B\}]$ and $[\{A\}\{C\}]$, $[\{B\}\{B\}]$ with $[\{B\}\{A\}]$ and $[\{B\}\{C\}]$, $[\{A\}\{C\}]$ with none and $[\{B\}\{C\}]$ with none. Thus:

C3	
Subsequence	**Count**
$[\{A, B\}\{A\}]$	1
$[\{A, B\}\{B\}]$	1
$[\{A, B\}\{C\}]$	3
$[\{B\}\{A, B\}]$	1
$[\{B\}\{A\}\{C\}]$	1
$[\{B\}\{B\}\{A\}]$	0
$[\{B\}\{B\}\{C\}]$	2

$\Longrightarrow$

F3	
Subsequence	**Count**
$[\{A, B\}\{C\}]$	3
$[\{B\}\{B\}\{C\}]$	2

And that's it because we cannot combine anything else. Thus, the result is F1, F2 and F3.

14.3 Sequential PAttern Discovery using Equivalence classes (SPADE) Algorithm

1. Transform de database into its vertical format, i.e., with SeqID, elemID inside sequence, and the element itself.
2. Construct the **ID-list** of 1-sequences, i.e., construct a table in which the elements are in the columns and in the cells we write all pairs $(seqId : elemID)$ in which the element in the column appears.
3. We count distinct seqID for each element. Those having more than *minsup* are kept and the rest are discarded.
4. For $k > 1$:
   (a) Contruct new k-candidates using the $(k-1)$-frequent subsequences (as in GSP). Prune when possible.
   (b) Construct the ID-list of k-sequences as $(seqID : elemID_1, ..., elemID_k)$ where $elemID_i$ is the element in $seqID$ in which the event i in the current subsequence appear.

(c) Count distinct seqID for each element. Those having more than *minsup* are kept and the rest are discarded.

Example 14.3. Let's repeat the example with SPADE. First, we transform the database to its vertical form:

Database	
Id	**Sequence**
1	$[\{A,B\}\,\{C\}\,\{A\}]$
2	$[\{A,B\}\,\{B\}\,\{C\}]$
3	$[\{B\}\,\{C\}\,\{D\}]$
4	$[\{B\}\,\{A,B\}\,\{C\}]$

$\Longrightarrow$

Vertical Database		
SeqID	**ElemID**	**Sequence**
1	1	$[\{A,B\}]$
1	2	$[\{C\}]$
1	3	$[\{A\}]$
2	1	$[\{A,B\}]$
2	2	$[\{B\}]$
2	3	$[\{C\}]$
3	1	$[\{B\}]$
3	2	$[\{C\}]$
3	3	$[\{D\}]$
4	1	$[\{B\}]$
4	2	$[\{A,B\}]$
4	3	$[\{C\}]$

Now, we construct the ID-list of 1-sequences:

	1-ID-List			
	$[\{A\}]$	$[\{B\}]$	$[\{C\}]$	$[\{D\}]$
	1:1	1:1	1:2	3:3
	1:3	2:1	2:3	
	2:1	2:2	3:2	
	4:2	3:1	4:3	
		4:1		
		4:2		
Count	3	4	4	1

$\Longrightarrow$

F1	
Subsequence	**Count**
$[\{A\}]$	3
$[\{B\}]$	4
$[\{C\}]$	4

Now the ID-list of 2-sequences, combining the 1-sequences:

	2-ID-List									
	$[\{A\}\,\{A\}]$	$[\{A\}\,\{B\}]$	$[\{A,B\}]$	$[\{B\}\,\{A\}]$	$[\{B\}\,\{B\}]$	$[\{A\}\,\{C\}]$	$[\{A,C\}]$	$[\{C\}\,\{A\}]$	$[\{C\}\,\{C\}]$	[{
	1:1,3	2:1,2	1:1,1	1:1,3	2:1,2	1:1,2		1:2,3		
			2:1,1	4:1,2	4:1,2	2:1,3				
			4:2,2			4:2,3				
Count	1	1	3	2	2	3	0	1	0	

So

F2	
Subsequence	**Count**
$[\{A,B\}]$	3
$[\{B\}\,\{A\}]$	2
$[\{B\}\,\{B\}]$	2
$[\{A\}\,\{C\}]$	3
$[\{B\}\,\{C\}]$	4

And now we do the 3-ID-List combining these:

	3-ID-List						
	$[\{A,B\}\{A\}]$	$[\{A,B\}\{B\}]$	$[\{A,B\}\{C\}]$	$[\{B\}\{A,B\}]$	$[\{B\}\{A\}\{C\}]$	$[\{B\}\{B\}\{A\}]$	$[\{B\}\{B\}\{C\}]$
	1:1,1,3	2:1,1,2	1:1,1,2	4:1,2,2	4:1,2,3		2:1,2,3
			2:1,1,3				4:1,2,3
			4:2,2,3				
Count	1	1	3	1	1	0	2

So

F3	
Subsequence	**Count**
$[\{A,B\}\{C\}]$	3
$[\{B\}\{B\}\{C\}]$	2

And the result is (obviously) the same we got with GSP.

14.4 PrefixSpan

1. Start by counting frequent 1-sequences, as in GSP.
2. PrefixSpan extends each frequent itemset recursively.
3. For each frequent 1-sequence S:
   (a) Project the database with S as a prefix, i.e., for each sequence, remove everything until the first occurence of S is found.
   (b) Count the possible expansions of S with one additional event at the end.
   (c) For frequent 2-sequences, repeat the same procedure. Until no new frequent k-sequences are discovered.

Example 14.4. We are going to do the same example again. The database is

Database	
Id	**Sequence**
1	$[\{A,B\}\{C\}\{A\}]$
2	$[\{A,B\}\{B\}\{C\}]$
3	$[\{B\}\{C\}\{D\}]$
4	$[\{B\}\{A,B\}\{C\}]$

with frequent 1-itemsets

F1	
Subsequence	**Count**
$[\{A\}]$	3
$[\{B\}]$	4
$[\{C\}]$	4

So we can start with A as a prefix. We project the database:

Database projected to $\{A\}$	
Id	**Sequence**
1	$[\{A,B\}\{C\}\{A\}]$
2	$[\{A,B\}\{B\}\{C\}]$
3	$[\{B\}\{C\}\{D\}]$
4	$[\{B\}\{A,B\}\{C\}]$

And we count possible 2-sequences starting with $\{A\}$:

C2 projected to $\{A\}$	
Subsequence	**Count**
$[\{A\}\{A\}]$	1
$[\{A\}\{B\}]$	1
$[\{A,B\}]$	3
$[\{A\}\{C\}]$	3
$[\{A,C\}]$	0

Now we select $\{A,B\}$ as prefix and count its possible extensions:

Database projected to $\{A,B\}$		**C3 projected to $\{A,B\}$**	
Id	**Sequence**	**Subsequence**	**Count**
1	$[\{A,B\}\{C\}\{A\}]$	$[\{A,B\}\{A\}]$	1
2	$[\{A,B\}\{B\}\{C\}]$	$[\{A,B\}\{B\}]$	1
4	$[\{A,B\}\{C\}]$	$[\{A,B\}\{C\}]$	3

Now we select $\{A,B\}\{C\}$ as prefix and again count its possible extensions (in this case the projected database only ahs one record, so there is no need to count):

Database projected to $\{A,B\}\{C\}$	
Id	**Sequence**
1	$[\{A,B\}\{C\}\{A\}]$

And with $\{A\}\{C\}$:

		C3 projected to $\{A\}\{C\}$	
Database projected to $\{A\}\{C\}$		**Subsequence**	**Count**
Id	**Sequence**	$\{A\}\{C,A\}$	0
1	$[\{A,B\}\{C\}\{A\}]$	$\{A\}\{C\}\{A\}$	1
2	$[\{A,B\}\{B\}\{C\}]$	$\{A\}\{C,B\}$	0
4	$[\{A,B\}\{C\}]$	$\{A\}\{C\}\{B\}$	0
		$\{A\}\{C\}\{C\}$	0

We would repeat the same with prefixes $\{B\}$ and $\{C\}$, and the result has to be the same as with the two previous algorithms.

14.5 Some comments

Algorithm	How	Time	Space
GSP	Candidate Generation	Counting against DB	Space alloc for candidate generation
SPADE	Candidate Generation	Counting against ID-Lists	Space alloc for the ID-Lists
PrefixSpan	No Candidate Generation (prefix-based generation)	Counting the projected DB	Space alloc for the projected DB, but there is one possible optimization using pointers

Part VI
Stream Data Mining

15 Stream data mining

A **data streaming** is a constant flow of data, which usually carries to much information for it to be feasible to be stored. Somehow, we need to be able to apply data mining algorithms in datasets whose records we are only able to see once. This might seem esoteric, but there are plenty of applications in which we find this kind of needs. For instance, in credit card transactions, in wearable sensors information, connected vehicles, Internet of Things,...
More precisely, the challenges that data streams possess are:

- **One pass constraint**: data size is assumed to be infinite. We cannot store everything to perform second passes through it. Say we want to cluster this type of data, we cannot apply an iterative approach like that of k-means.
- **Drift**: data evolves over time, as well as its statistical properties. This means that a model that works well today, could work poorly tomorrow. We need to be able to adapt to these changes.
- **Resource constraints**: we are constrained to the arrival rate of the data, which can be variable and sometimes it can has huge peaks where data is coming faster than expected. This implies that the algorithms need to be fast enough to not lose valuable data because of processing.
- **Massive domain**: some data attributes might have a large values of distinct values.

15.1 Bloom filter

The bloom filter is an idea thought to answer the question:

'Has this incoming element ever ocurred in the data stream before?'

The idea is to be able to summarize the past of the data stream to be able to answer this question with a reasonable level of confidence. A **bloom filter** is a data structure that consists of:

- A binary array of length m.
- w independent hash functions $h_1, ..., h_w$.

And the algorithm developed to initialize this data structure is the one described in Algorithm 11. Basically, each element is hashed using all the hash functions, then the correspondent elements in the array are set to True.

Algorithm 11 BloomConstruct(Stream S, Size m, Number of hash w)

```
B = array(length=m, type=bool, default=False)
repeat
        receive next element x in S
        for i = 1..w
                idx = hi(x)
                B[idx] = True
until S ends
return B
```

Now, when a new element comes from the stream, $x \in S$, we would compute $h_1(x), ..., h_w(x)$. If all the correspondent elements in the bloom filter are True, then we are confident (if the hash functions are well done) that the element has already been seen before. If some of them is False, we know for sure that the element has not been seen before, and we update the filter.

Example 15.1. Suppose we have the stream $[3, 8, 15, 3, 5]$, $m = 10$, $h_1(k) = k \mod 10$ and $h_2(k) = (k + 7) \mod 10$. The procedure would be:

1. Start with all-False bloom filter:

F	F	F	F	F	F	F	F	F	F

2. 3 enters: $h_1(3) = 3$ and $h_2(3) = 0$. Both F, so 3 has never been seen before. Update B:

T	F	F	T	F	F	F	F	F	F

3. 8 enters: $h_1(8) = 8$ and $h_2(8) = 5$. Both F, so 8 has never been seen before. Update B:

T	F	F	T	F	T	F	F	T	F

4. 15 enters: $h_1(15) = 5$ and $h_2(15) = 2$. The first is T, but the second is F, so 15 has never been seen before. Update B:

T	F	T	T	F	T	F	F	T	F

5. 3 enters: $h_1(3) = 3$ and $h_2(3) = 0$. Both T, so we assume 3 has indeed been seen before. In this case we are right! B is not updated.

6. 5 enters: $h_1(5) = 5$ and $h_2(5) = 2$. Both T, so we assume 5 has indeed been seen before. In this case we are wrong... it is a false positive.

Remark 15.1. Note how little by little the array gets populated by True values, making false positives increasingly likely.

15.2 Count-Min Sketch

In this case, we are interested in answering the question:

'How many times has this incoming element appeared in the data stream before?'

Again, the idea is to be able to make a summary of the past data that enables us to answer the question. A **count-min sketch** is a data structure which consists of:

- A set of w different numeric arrays, each of length m.
- w independent hash functions $h_1, ..., h_w$.

And the algorithm developed to initialize this data structure is the one described in Algorithm . Basically, each element is hashed using all the hash functions, then the correspondent elements in the array are augmented by 1 unit. There are w rows to make the values harder to collide (two elements x, y collide only if $h_i(x) = h_i(y)$, for the same h_i). To get the number of times an element has come, we get the minimum of the cells accessed, because the minimum is an upper bound for the number of times the element has been seen.

Example 15.2. Suppose we have the stream $[3, 8, 15, 3, 5]$, $m = 10$, $h_1(k) = k \mod 10$ and $h_2(k) = (2k) \mod 10$. The procedure would be:

1. Start with all-0 count-min sketch:

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

2. 3 enters: $h_1(3) = 3$ and $h_2(3) = 6$. Both 0, so 3 has been seen 0 times before. Update CM:

0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0

Algorithm 12 CountMinConstruct(Stream S, Width w, Height m)

```
CM = Matrix(nrows=w, ncols=m, type=int, default=0)
repeat
        receive next element x in S
        for i=1..w
                idx = hi(x)
                CM[i,idx] += 1
until S ends
return CM
```

3. 8 enters: $h_1(8) = 8$ and $h_2(8) = 6$. One cell is 1 and the other is 0, so 8 has never been seen before. Update CM:

0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	2	0	0	0

4. 15 enters: $h_1(15) = 5$ and $h_2(15) = 0$. Both 0, so 15 has never been seen before. Update CM:

0	0	0	1	0	1	0	0	1	0
1	0	0	0	0	0	2	0	0	0

5. 3 enters: $h_1(3) = 3$ and $h_2(3) = 6$. One is 2 and other is 1, so it has been seen once before. Update CM:

0	0	0	2	0	1	0	0	1	0
1	0	0	0	0	0	3	0	0	0

6. 5 enters: $h_1(5) = 5$ and $h_2(5) = 0$. Both 1, so we assume 5 has indeed been seen before once. In this case we are wrong... it is a false positive. The table would be updated again:

0	0	0	2	0	2	0	0	1	0
2	0	0	0	0	0	3	0	0	0

Remark 15.2. Note that if 15 went in again, we would fail again, and in fact from this point on we will always fail with the count of 15s and 5s seen.

15.3 Flajolet-Martin algorithm

The Flajolet-Martin algorithm aims at answering the question

'How many distinct elements appeared in the data stream before?'

The intuitive idea is as follows: if the hash function distribute evenly the numbers in the range of the function, then we can assume that the probability of getting a particular value after hashing is $\frac{1}{|range|}$. Thus, if we take the binary representation of the output, we can assume that the last digit is 0 with probability $\frac{1}{2}$ because it has the same chance of being 0 or 1. Now, the output having the last two digits as 0 would occur approximately 1 out of 4 times, and so on... The probability that the last k digits are 0 is $\frac{1}{2^k}$, so if we see a value with k trailing 0, the expected amount of seen records is 2^k.
The algorithm works as follows:

1. Given a data stream S, use a hash function $h : S \to [0, 2^L - 1] \cap \mathbb{Z}$, where L is such that $2^L > \#distinct\ elements$. L is usually chosen to be 64.
2. For each incoming record x, take the binary representation of $h(x)$.
3. Count the number of trailing zeros in $h(x)_2$.

4. Keep a variable R_{max} with the maximum number of zeros found until now.
5. The expected maximum number of trailing zeros over all stream elements is

$$E\left[R_{max}\right] = \log_2\left(0.77351n\right),$$

where n is the amount of distinct values. So, if we want to know how many distinct values we have seen until now, we would estimate

$$n \sim \frac{2^{R_{max}}}{0.77351}.$$

Example 15.3. Suppose we have the stream $[3, 8, 15, 3, 5]$, $h\left(x\right) = \left(7x + 5\right) \mod 32$.

1. 3 enters: $h\left(3\right) = 26 = 11010$. $R_{max} = 1$.
2. 8 enters: $h\left(8\right) = 19 = 10011$. $R_{max} = 1$.
3. 15 enters: $h\left(15\right) = 14 = 01110$. $R_{max} = 1$.
4. 3 enters. $R_{max} = 1$.
5. 5 enters: $h\left(5\right) = 8 = 01000$. $R_{max} = 3$.

Thus, the estimation is

$$n \sim \frac{2^3}{0.77351} = 10.3,$$

and the real value is 4. Note, nonetheless, that as we are working with probabilities, the results are not expected to be accurate when there are very few values.

Improvements to Flajolet-Martin

Improvement 1) It can happen that one of the first values has lots of trailing zeros, ruining the algorithm. A solution to avoid this is using several hash functions $h_1, ..., h_w$ and keep the maximum number of trailing zeros for each of them $R_{max,1}, ..., R_{max,w}$. At the end, we would obtain R_{max} as the average of all these.

Improvement 2) It is also possible to have several points in which the records are taken, and thus we would like to synchronize the results from every of them. The solution is to treat all these points as if they were only one. Let's explain this a little bit. Imagine we have m measure points, where m streams are measured, one stream per measure point. Then, we would have m values $R_{max,i}$ for $i = 1, ..., m$ and we want to obtain the combined R_{max}. The idea is that, if we think of all the m streams as a single stream which has been divided, then we would just take R_{max} as the maximum number of trailing zeros observed in the stream. Thus, when it is divided, if we know the maximum number of trailing zeros in each of the substreams, we also know the maximum number of trailing zeros in the whole stream: the maximum among the substreams! Thus, we would take

$$R_{max} = \max_{i=1,...,m}\left\{R_i\right\}.$$

Each of the substreams would work exactly as explained before (maybe with the Improvement 1 implemented).

15.4 Hyperloglog

Hyperloglog is a generalization of Flajolet-Martin, which tries to improve the predictions to answer the same question of how many different values have been seen in the stream before.
The idea is that we use 2^k buckets to count the trailing zeros observed in the records observed. For an incoming record, x, we compute $h\left(x\right)$ and translate it to binary form. Then, the bucket in which it will count is the bucket numbered with the first k bits of $h\left(x\right)_2$. In each bucket, we would count the maximum number of trailing zeros in records classified into that bucket. At the end, as explained by Flajolet et al. in [2], the estimation is computed as

$$n \sim k\frac{HM_{i=1}^{k}\left(2^{R_{max,i}}\right)}{0.77351},$$

where k is the number of buckets, and $HM_{i=1}^{k}\left(x_i\right) = \frac{k}{\frac{1}{x_1}+...+\frac{1}{x_k}}$ is the harmonic mean.

Example 15.4. Suppose we have the stream $[3, 8, 15, 3, 5]$, $h(x) = (7x + 5) \mod 32$ and there are two buckets.

1. 3 enters: $h(3) = 26 = 11010$. $Bucket = 1 \to R_1 = 1$.
2. 8 enters: $h(8) = 19 = 10011$. $Bucket = 1 \to R_1 = 1$.
3. 15 enters: $h(15) = 14 = 01110$. $Bucket = 0 \to R_0 = 1$.
4. 3 enters. $Bucket = 1 \to R_1 = 1$.
5. 5 enters: $h(5) = 8 = 01000$. $Bucket = 0 \to R_0 = 3$.

Thus, the estimation is

$$n \sim 2\frac{\frac{2}{\frac{1}{3}+\frac{1}{1}}}{0.77351} = 3.88.$$

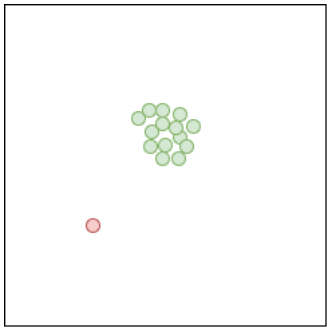


Figure 3: A global outlier.

Part VII
Outlier mining

16 Outlier Mining

Definition 16.1. An **outlier** is a data object that deviates significantly from the normal objects as if it were generated by a different underlying mechanism.

Remark 16.1. Note that an outlier is not the same as noise in the data. Noise is due to random errors or the variance in a measured variable and for a good outlier analysis, it is required that noisy records are removed first.
Outlier are far more interesting than noise, because they are generated differently from the usual data.
As we saw in stream data mining, data can vary over time, and in the early stages of a change process, the new records would be seen as outliers from the past ones. But little by little we would notice that they are not outliers, they are the result of a change in the underlying process. Thus, outlier detection is also a part of the novelty detection process.

Remark 16.2. Some use cases are:

- Credit card fraud detection: the normal records are not fraudulent, and they add up to a great majority of the records. Thus, the frauds are outliers which are generated differently.
- Medical analysis: the outliers are the few people that suffer an illness.

16.1 Types of outliers

- **Global outlier**: is a record that significantly deviates from the rest of the data set. It can be found clustering the data and defining an appropriate measurement of deviation.
- **Contextual outlier**: is a record that deviates significantly based on a selected context, i.e., a subset of the data. Usually, attributes can be categorized into:
  - Contextual attributes: which defines the context of the record, e.g. the time and location.
  - Behavioral attributes: these are the proper measures of the records, which are used in outlier evaluation, e.g. temperature.

  This way, if the context is Brussels in December, a temperature of 0 degrees would probably not be an outlier. But the same temperature in Dominican Republic in June would be an outlier.

  The main problem in this case is how to define meaningful context.

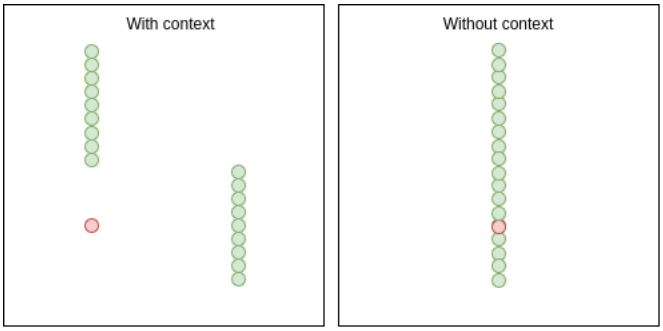


Figure 4: A context outlier.

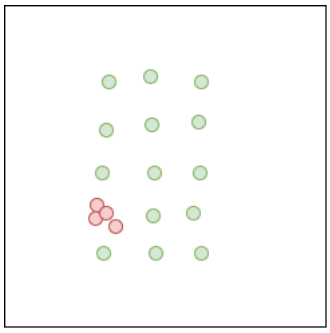


Figure 5: Collective outliers.

- **Collective outliers**: a subset of the data records collectively deviates significantly from the whole data set, but the individual data records might not be outliers themselves. For example, in the lottery, someone has to win, so when we look at individuals winning a lottery prize we would not see anything strange. But if the same person wins several editions of the lottery, then it would be an outlier.

16.2 Challenges of outlier detection

- It is hard to enumerate all possible normal behaviors in an application and the boundary that separates normal data from outlier data is often in a gray area which is hard to identify.
- The choice of distance measures among objects and the relationship between objects is application-dependent, so there is no one-fits-all solution.
- As mentioned before, noice can blur the distinction between normal objects and outliers because the algorithms can sometimes confuse noise with outliers.
- The understandability of the outliers is also hard to provide. It is not easy to understand how the detected outliers are produced in contrast to normal records. Usually, one thing we can do is estimate the unlikelihood of the record being generated by a normal mechanism (hypotheses testing).

16.3 Supervised methods for outlier detection

We can try to model outlier detection as a classification problem, in which the samples are examined and classified by domain experts and are later used for training and testing models.

16.3.1 One-Class model

The idea of the one-class model is to train a classification model that can distinguish normal data from outliers. For this, it requires many abnormal samples, as many as possible. The functioning is to learn the decision boundary of the normal class. Then, all samples that lie outside this boundary are labeled as outliers.
It poses the **advantage** that can detect new outliers that are different from past outliers.

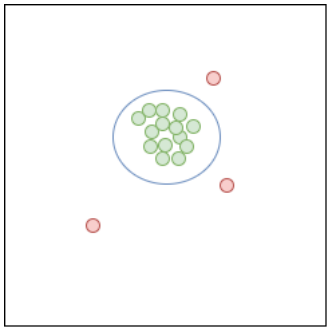


Figure 6: Basic diagram of a one-class model.

It is possible to extend the model to a **multi-class model**, in which the normal objects might be also classified into multiple classes.

16.4 Unsupervised methods for outlier detection

If we assume that the normal objects are somehow clusered into multiple groups, each of them having some distinctive features, we can expect outliers to be far away from any group of normal records.
This approach posses the **drawback** that it is very hard to detect collective outliers effectively.

16.4.1 Proximity-based methods

In this case, a record is considered an outlier if the nearest neighbors of the objects are far away. The effectiveness of these methods relies on the proximity measure used and it is often hard to find groups of outliers that are close to each other.

Distance-based outlier detection

Let $r > 0$ be a distance threshold and $\tau \in (0, 1]$ be a fraction threshold. An object o is a $DB\left(r, \tau\right) - outlier$ if

$$\frac{card\left\{o' | dist\left(o, o'\right) \leq r\right\}}{card\left(D\right)} \leq \tau.$$

So we count how many objects are within a distance of r and if these account to less than the predefined fraction τ, then o is considered an outlier.
If we do it just like this, we would test each object against the whole data set one by one. This can be very costly for big datasets, and can be improved with the **CELL method**, which is a grid-based method. In this case, the data space is partitioned into a multidimensional grid, in which is cell is a hyper cube with diagonal length $\frac{r}{2}$. This allows us to improve the efficiency of the method by using two pruning rules:

1. **Definitions**: Let C be the cell whose objects we want to assess:
   (a) The **level-1** are the cells adjacent to cell C.
   (b) The **level-2** are the cells which have some point less than r distance from C.
2. **Cell** C: it is obvious that all objects in cell C are less than distance r from each other. Call $a = card\left(C\right)$.
3. **Level-1 cell pruning rule**: let $b_1 = card\left(level - 1\right)$. If $a + b_1 > \lceil \tau n \rceil$, then every object o in C is not a $DB\left(r, \tau\right)$-outlier because all objects in $C \cup level - 1$ are in the r-neighborhood of o and there are at least $\lceil \tau n \rceil$.
4. **Level-2 cell pruning rule**: let $b_2 = card\left(level - 2\right)$. If $a + b_1 + b_2 < \lceil \tau n \rceil + 1$, then all objects in C are $DB\left(r, \tau\right)$-outliers.

16.4.2 Density-based methods

The idea in this case is that the density around an outlier object is significantly different from the density around its neighbors. Thus, we can use the relative density of an object against its neighbors as the indicator of the degree of the object being an outlier.
For this, we define the k-distance of an object o as the distance between o and its k^{th} nearest neighbor, and the k-distance neighbourhood of o as

$$N_k(o) = \{o' | o' \in D, dist(o, o') \leq dist_k(o)\}.$$

Note that this set can have more than k elements because multiple elements can have identical distance to o. Thus the measure $\frac{k}{card(N_k(o))}$ is an indicator of the density around the point o, which can be compared with the rest to decide if o is an outlier or not.

16.4.3 Clustering-based methods

The idea now is to cluster the data and classify an object as an outlier if:

1. It does not belong to any cluster.
2. There is a large distance between the object and its closest cluster.
3. It belongs to a small or sparse cluster.

This approach presents several advantages:

- It can detect outliers without the need to label the data.
- It works for many datatypes.
- Clusters can be regarded as summaries of the data.
- The detection is fast once the clusters are obtained, because we only need to compare any object against the clusters.

But it also has some drawbacks:

- The effectiveness depends on the cluster method used.
- The cost of the clustering can be high. This can be partially fixes using fixed-width clustering:
  - A point is assigned to a cluster if the center of the cluster is within a predefined distance threshold from the point.
  - If the point cannot be assigned to any existing cluster, a new cluster is created and the distance threshold may be learned from the training data under certain conditions.

References

[1] Charu C. Aggarwal. *Data Mining.* Springer International Publishing, 2015.

[2] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

[3] Mahmoud Sakr. Infoh423 data mining. Lecture Notes.

[4] Ian Witten, Eibe Frank, and Mark Hall. *Data Mining: Practical Machine Learning Tools and Techniques.* Elsevier, 2011.