# 7CCSMDM1
# Data Mining

If you spot anything wrong, please let me know!
discord: @insertish
izzy · insrt.uk

## Contents

# 1. Introduction to Data Mining

The goal of data mining is to extract information from data, i.e. understand it and take advantage.

Computers allow generating, managing, processing, and communicating data and information.

- Data is raw, unorganised facts which are not priori useful.
- If data is processed, organised, structured, and meaningfully presented in some context, then it becomes **information**.

Information is *hidden* in data and understanding tends to decrease as volume of data increases.

**Data Mining** is the process of discovering patterns in data

- it must be automated
- patterns must be meaningful and useful, i.e. lead to benefit/inform decisions
- works on existing data, i.e. data that has already been generated

Finding patterns in data involves:

1. Identifying patterns
2. Validating patterns
3. Using patterns for predictions

**Real-world Applications**

- *PageRank*: assign measures to web pages, based on search query relevance
- *Email filtering*: classify new messages as spam/ham
- *Online advertising*: find users with similar purchases, etc
- *Social media*: identify users with similar preference
- *Marketing and Sales*: identiffy customers likely to defect and fight churn; market basket analysis for personalised offers; etc
- *Risk*: statistical calculation of bank loan default risk; anticipated job candidate performance in recruitment
- *Image data*: detect oil spills/deforestation in satellite images; currency recognition in automated payment machines; face recognition for surveillance
- *Engineering*: power demand forceasting for suppliers; failure prediction for machine maintenance in manufacturing

## 1.1. Data Sets

The following data sets are used in this module:

- Contact Lens Data [WFH3, Table 1.1]
- Weather Data [WFH3, Table 1.2], see Table 2
- Weather Data with Some Numeric Attributes [WFH3, Table 1.3]
- CPU Performance Data Set [WFH3, Table 1.5]
- Family Tree [WFH3, Figure 2.1]
- Attribute-Relation File Format (ARFF) [WFH3, Figure 2.2]

*Sourced from I. H. Witten, E. Frank, M. A. Hall. Data Mining: Practical Machine Learning Tools and Techniques (3rd Ed.), Morgan Kaufmann, 2011.*

Data has a range of structure to it:

- *Structured data* is typically stored in relational databases: sales/transactional data, customer database, healthcare records
- *Semi-structured data* does not have a strict format, but have some organisation: xml, json, emails
- *Unstructed data* lack predefined semantic structure: text data, images, audio files

An **attribute** is (feature/column) used to characterise each data set entry; may depend on each other

An **instance** is (example/row) set of values, one for each attribute; typically considered independent but there may be relationships between instances

**Attribute Types**
- **Numeric** is continuous or discrete with well-defined distance between values
- **Nominal** is categorical (qualitative data)
- **Dichotomous** is binary; boolean; yes/no
- **Ordinal** is ordered categorical but without well-defined distance, e.g. poor, reasonable, good, excellent (qualitative data)
- **Interval** is ordered categorical, but measured in fixed units, e.g. cool, mild, hot temperatures

> **Note**
>
> This module includes *simple data sets* appropriate for learning. Practicals will use wide range of data sets from online sources. Often real-world data sets contain thousands/millions of entries, are incomplete, are noisy, incorporate randomness, and are sparse.

A signficiant part of working as a data scientist is *data preparation* which may require assembly, integration, cleaning, and transformation of data.

**Feature Engineering** is the process of transforming raw data by selecting the most suitable attributes for the data mining problem to be solved

Patterns allow making non-trivial predictions for new data, could be derived from:
- A black box (hidden structure)
- Transparent data (visible structure)

Structural patterns capture and explain data aspects in an explicit way and can be used for better-informed decisions. e.g. rules in form if-then-else

Patterns can be captured by different types of models:
- Linear equations
- Clusters
- Tree structures

**Example of classification from weather dataset Table 2**

Attribute values predict the label:

```py
if outlook == "sunny" and humidity == "high":
  play = "no"
```

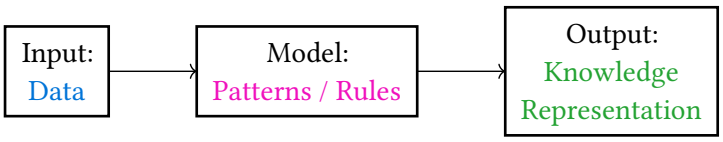
The input is four attributes:
- outlook = { sunny, overcast, rainy }
- temperature = { hot, mild, cold }
- humidity = { high, normal }
- windy = { yes, no }

The output is a decision, i.e. one label/class:
- play = { yes, no }

There are 14 cases present in the data set (36 possible cases – conditions), a rule may use or more attributes to make the right decision but the rules obtained may not be good.

Data mining aims to construct models from the data.
If the data set is complete, then the rules produce 100% correct predictions.
Often data sets are incomplete, and produce incorrect predictions as information is missing.

Given a data set, we aim to construct models expressing these patterns using some form of knowledge representation:



## 1.2. Conceptual Framework

**Classification** is modelling relationships between data elements to predict classes or labels
- Data is classified, e.g. people can be labelled as Covid positive/negative based on symptoms
- Models the way attributes determine the class of instances
- This is a *supervised learning* task because it uses already classified instances for further predictions

**Regression / Numeric Prediction** is modelling relationships between data elements to predict numeric quantities
- Models ways that attributes determine a numeric value
- Variant of classification without discrete classes
- *Supervised learning* task
- Often, produced model is more interesting than predicted values, e.g. what attributes affect preds.

**Clustering** is modelling relationships of instances to group them so that instances in the same group are similar than instances in different groups
- By labelling clusters, we may use them in meaningful ways
- *Unsupervised learning* task as data is not labelled

**Association** is modelling relationships between attributes
- No specific class or label
- May examine any subset of attributes to predict any other disjoint subset attributes
- Usually involve only nominal data

### 1.2.1. Process of Data Mining

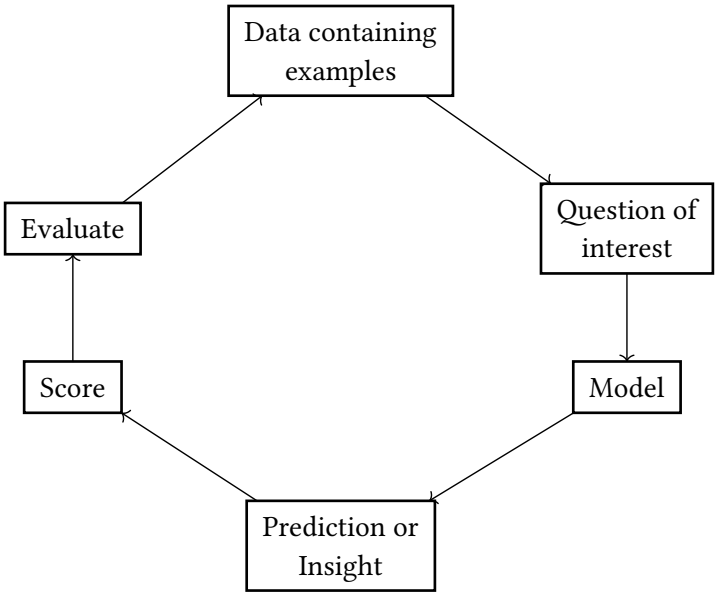



Figure 1: Typical Data Mining Process

**Typical Process**

1. **Objective Specification**: identify data mining problem type
   - Supervised learning: there is a target attribute
     - *Nominal data*: use classification
     - *Numeric data*: use prediction
   - Unsupervised learning: no target attribute
     - Cluster instances into groups of similarity
     - Or find attribute correlations/associations
   - … other data mining tasks may be applicable
2. **Data Exploration**: visualise data, e.g. histograms/scatter plots

   Confirm the objective can be achieved with the data set
3. **Data Cleaning**: fix any problems with the data
   - Confirm there is enough data (broad and deep)
     Sparse data means data mining is less effective.
     Rule of thumb is that more data is better.
     Large data sets can become problematic when target variable appears in extremely rare patterns or when the model building is very resource consuming.
   - Check for imprecise or missing values
   - Verify data set is representative and not biased
4. **Model Building**: select most appropriate model for the data
5. **Model Evaluation**: assess whether the model achieves set out goals

   tl;dr measure accuracy, split existing/new data into training, validation, test sets, check for overfitting, and use other measures of success.
6. **Repeat**: iteration to improve the model

# 2. Regression & Classification (Linear & Non-linear Models)

**Supervised Learning** is where we find a model/rule/function/concept definition that takes attributes and predicts labels or numeric quantities for data from a given training data set where each instance is associated with a set of attributes (input) and a class/label/numeric variable (output)

Classification and regression are supervised learning tasks.
- If the output is nominal, then the process is typically classification.
- If the output is numeric, then the process is usually regression or numeric prediction.

**Classification** is modelling relationships between data elements to predict classes or labels

A **linear classification model** is used for classification problems by drawing decision boundaries between regions, e.g. sets of points belonging to the same class

We **score a classification model** by determining for each instance whether the predicted label $\hat{y}$ matches the observed label $y$. If there is a match, we increment the score:

$$\text{score} = \sum \{1 \mid \hat{y}_i \in \text{predicted}, \hat{y}_i = y_i\}$$

We compute the **error rate for a classification model** by first scoring with respect to each instance of the data set:

$$\text{error rate} = 1 - \frac{1}{m} \sum_{i=1}^{m} \text{score}_i$$

**Regression / Numeric Prediction** is modelling relationships between data elements to predict numeric quantities

We **score a regression model** by determining for each instance, the distance between the predicted value $\hat{y}_i$ and the observed value $y_i$, i.e. compute residual, i.e. $(y_i - \hat{y}_i)^2$.

We compute the **error rate for a regression model** by scoring all instances and adding these scores. A common approach is the **sum of squared errors**:

$$\text{error rate} = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

**Building Classification/Regression Models**
- Training Step
  1. Select the training set, i.e. portion of data set
  2. Initialise model parameters $\boldsymbol{w} = (w_0, w_1, ..., w_n)$
  3. Apply model to all instances of the training set
  4. Compute the error rate of the model
  5. Adjust the parameters to obtain a model with lower error
  6. Repeat from step 3 until error rate is low enough
  7. Output model
- Evaluation Step
  1. Select the test set
  2. Apply the model to all instances of the test set
  3. Compute the error rate
  4. Output evaluation error

*Questions raised when building models*

- Data set splits: how to divide the data set into training and testing?
- Model structure: what model should be used, how many components, which parameters?
- Model building: how to init/adjust parameters, how to compute score, when to stop?

Jump to Section 3.3 for cross validation.

## 2.1. Linear Regression

*Preamble*

- 'All models are wrong, but some are useful' — George Box
- Model building in DM is a data-driven task
- Important goal is identifying relationships in data (correlations)
- Correlations may or may not correspond to real-world explanation, phenomenon, process, etc; correlation and causation are different.
- https://tylervigen.com/spurious-scholar

**Linear Regression Equation**

$$\begin{aligned}\hat{y}_i &= \boldsymbol{x}_i \boldsymbol{w} \\ &= w_0 + \sum_{j=1}^{n} w_j x_{i,j}\end{aligned}$$

Where $\hat{y}_i$ is the predicted value for instance $i$, $\boldsymbol{x}_i = \{1, x_1, ..., x_n\}$ is a vector of variables, and $\boldsymbol{w} = \{w_0, w_1, ..., w_n\}$ is a vector of model parameters.

Linear regression is the process of finding coefficients $\boldsymbol{w}$ of a function $\hat{y}$ that best approximate the relationships between variables $\boldsymbol{x}$ and output $y$. Our *objective* is to determine the values of the parameters $\boldsymbol{w}$ so that the error (usually sum or mean-squared differences) is minimised.

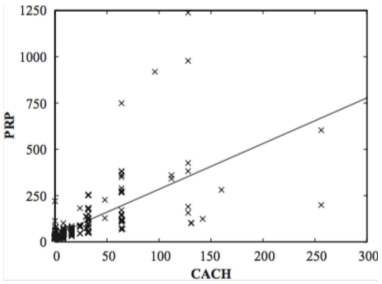



Figure 2: Linear regression models are often visualised as 2D scatter plots showing the regression line that best fits the data.

e.g. $\hat{y}_i = w_0 + w_1 \cdot x_i$

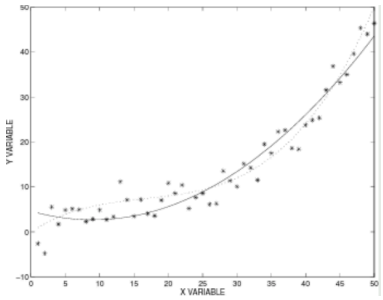



Figure 3: A model may have non-linear variables and linear parameters.

$$\hat{y}_i = w_0 + w_1 \cdot x_i + w_2 \cdot (x_i)^2$$

We may also have a **piecewise linear model** which uses multiple line segments to approximate.

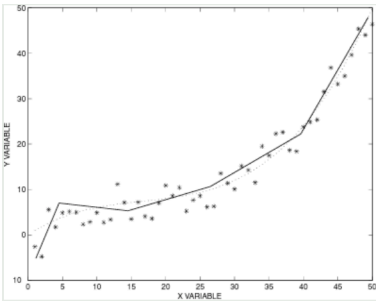



Figure 4: A piecewise model

**Overfitting** is a situation where the model predicts the training data well, but does not generalise to new data; often a danger with high model complexity

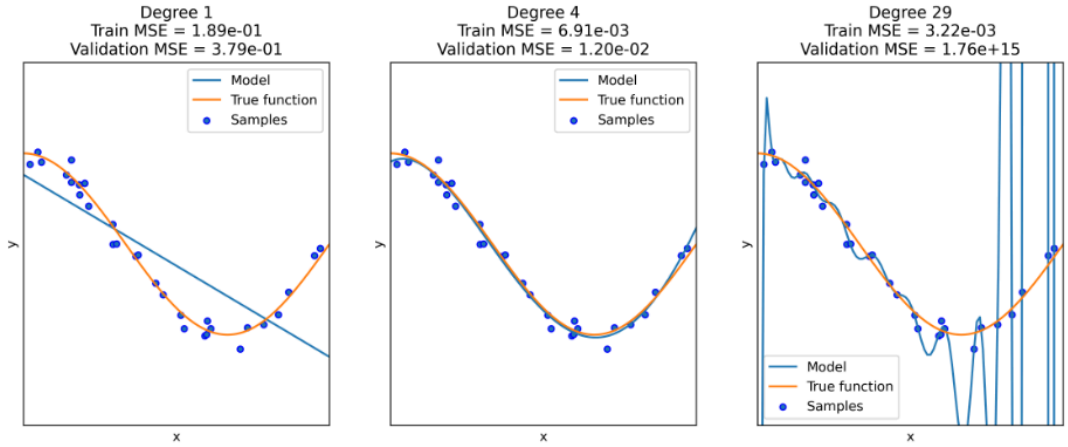



Figure 5: Evaluation samples for trained polynomial of varying degrees.

### 2.1.1. Training / Fitting Linear Regression

**Least Squares Linear Regression**: compute model by minimising the mean of squared errors:

$$S(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- Select parameters $\boldsymbol{w}$ that minimise $S(w)$
- For each instance $i$, the error/residual is the difference $y_i - \hat{y}_i$.
  Absolute difference: $|\ y_j - \hat{y}_j\ |$
  Squared difference: $\left(y_j - \hat{y}_j\right)^2$
- We say $S(w)$ is the objective function
- Approaches to solve the minimisation problem:
  - *gradient descent* which performs iterative search
  - *ordinary least squares* which computes a solution analytically with matrix inversion

**Gradient Descent** is modifying parameter values to make $S(\boldsymbol{w})$ by moving to the stepest descent direction

**Learning Rate** is a hyperparameter $\alpha$ which is a constant and controls the size of weight updates during gradient descent. If it is too large, then gradient descent may overstep the point $\boldsymbol{w}^*$ minimising $S(\cdot)$ but if it is too small it may need too many iterations to terminate

A sample termination criterion is: (where $\boldsymbol{w}^{(k)}$ are the weights in iteration $k$)

$$|\ S(\boldsymbol{w}^{(k)} - S\big(\boldsymbol{w}^{(k+1)}\big)\ | < \varepsilon$$

**Off-line, batch, multi-pass gradient descent** is where we follow gradient through computation over all instances, multiple times. (convergence is assessed after all instances have been examined)

```
w ← {0, ...} initialise w randomly or to zero
repeat until convergence is achieved
    for i ∈ {1, ..., m}:
        for j ∈ {1, ..., n}:
            w_j ← w_j + α(y_i − ŷ_i)x_{i,j}
until S(w) is not significantly modified
```

**On-line, single-scan, stochastic gradient descent** is approximating gradient through computation for each instance, one at a time. (convergence assessed after each individual instance)

```
w ← {0, ...} initialise w randomly or to zero
for i ∈ {1, ..., m}:
    repeat until convergence is achieved
        for j ∈ {1, ..., n}:
            w_j ← w_j + α(y_i − ŷ_i)x_{i,j}
    until S(w) is not significantly modified
```

This is referred to as stochastic as the gradient is approximated after each instance with possible errors so the guesses are referred to as stochastic. Given not all data is available, we refer to the algorithm as an online approximation.

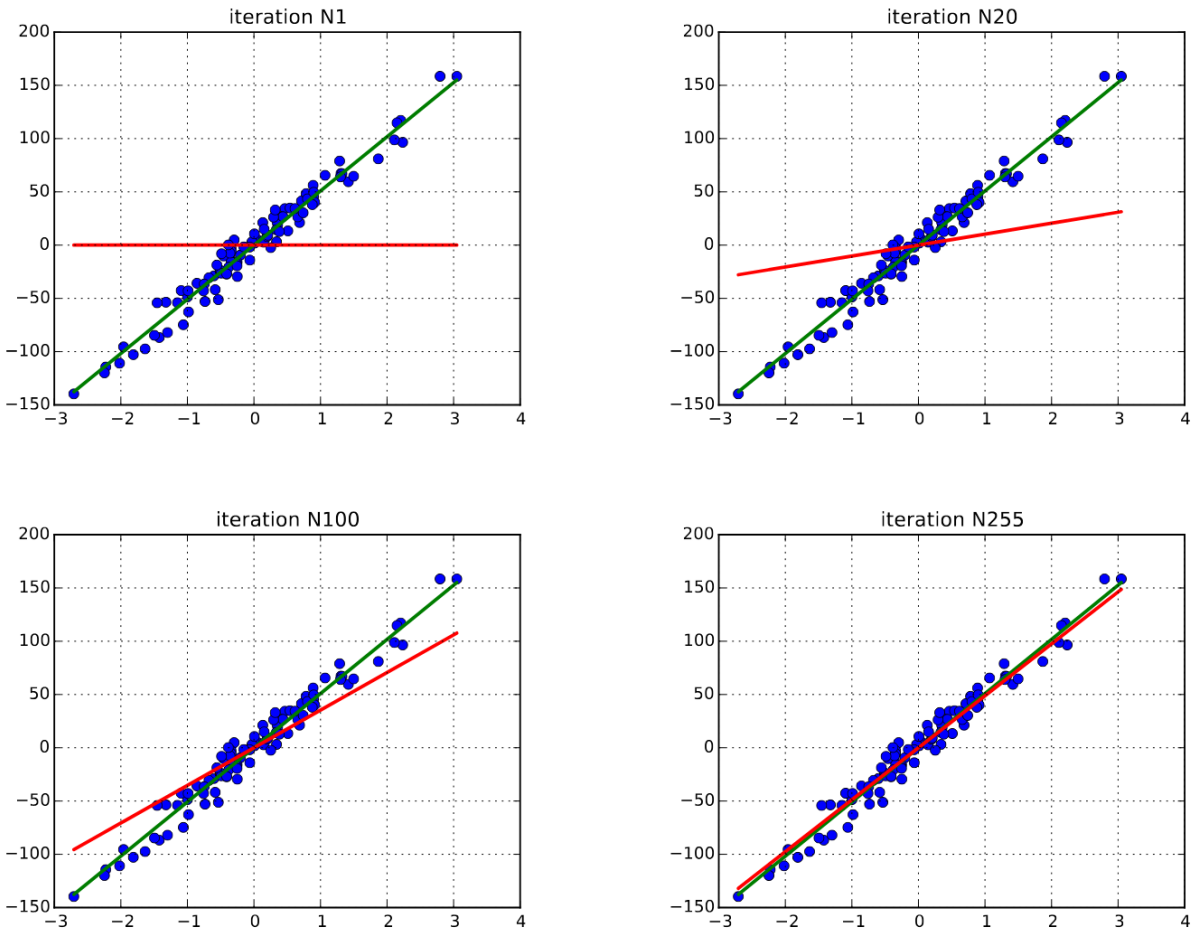



Figure 6: Gradient descent for least squares linear regression

## 2.2. Logistic Regression

Linear regression models may produce good models in practice but are often problematic, e.g. may need to be modified for each new instance, may produce arbitrary numeric values.

**Logistic Regression** is predicting the probability an instance belongs to a class by approximating probability using a logistic function

### 2.2.1. Using linear/logistic regression for classification

Let's assume a binary classifier (two classes, $\{0, 1\}$) where we build a best-fit linear regression model. For each instance $i$, predict 1 if $\hat{y}_i > 0.5$ else 0.

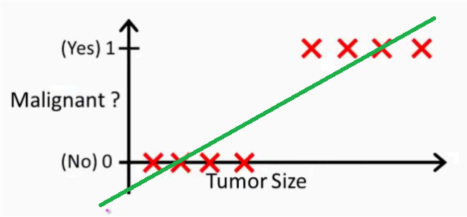



Figure 7: Example model

We may find good linear regression classification models in practice, but they are often problematic:
- May need to be modified for every new instance.
- May produce arbitrary numeric values.

It is more meaningful to predict the probability that an instance belongs to a class!

Logistic regression replaces the original target variable $\Pr(1 \mid x_1, ..., x_n)$, which cannot be approximated, with:

$$\log\left(\frac{\Pr(1|x_1, ...x_n)}{1 - \Pr(1|x_1, ..., x_n)}\right)$$

where the transformed variable is approximated with a linear function, $w_0 + w_1x_1 + ... + w_nx_n$.

> **Note**
>
> The estimate for $\Pr(1|x_1, ..., x_n)$ is given by:
>
> $$\hat{y} = \frac{1}{1 + \exp\left(-\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)\right)}$$
>
> Value is always between 0 and 1. Goal is to compute weights that fit training data well.
> We can use gradient descent to maximise log-likelihood.

To make a prediction, we draw a decision boundary:
- $\Pr(1|x_1, ..., x_n) = 0$ (equivalent to $w_0 + w_1x_1 + ... + w_nx_n = 0$)
- Therefore we may predict 1 if $w_0 + w_1x_1 + ... + w_nx_n \geq 0$ else 0.

> **Note**
>
> About generalising to multiple classes:
> - Performing logistic regression for each class independently may result in probability estimates that do not sum to 1.
> - A proper probability distribution yields a joint optimisation problem which we can solve for.

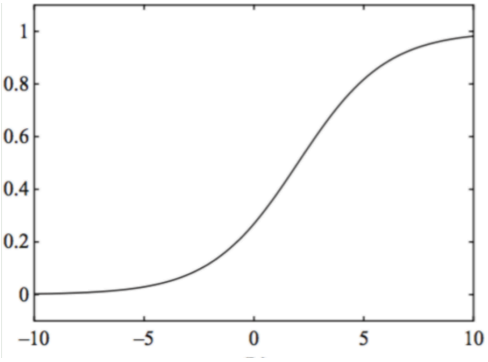
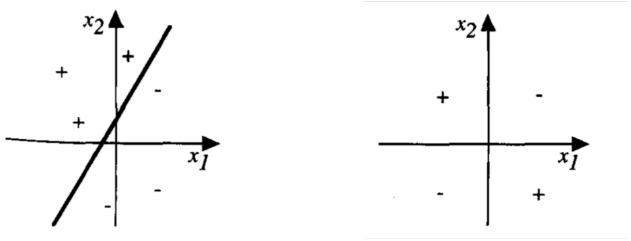



Figure 8: binary classification example with two weights $w_0 = -1.25, w_1 = 0.5$

The horizontal axis is $x_1$.
The vertical axis is $\Pr(1|x_1)$.

## 2.3. Perceptrons

**Linearly Separable Data** is can be separated into two classes using a hyperplane



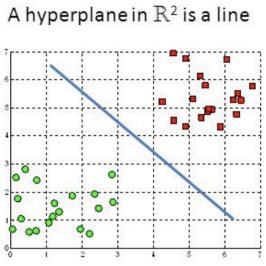

**Hyperplane Equation**

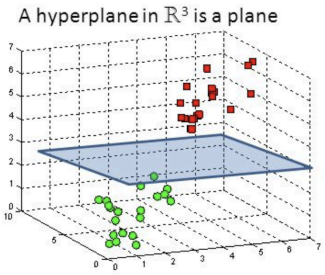
$$\boldsymbol{w} \cdot \boldsymbol{x} = 0$$

Where $\boldsymbol{w} = (w_0, w_1, ..., w_n)$
Where $\boldsymbol{x} = (1, x_1, ..., x_n)$
Where $w_0$ is the bias weight

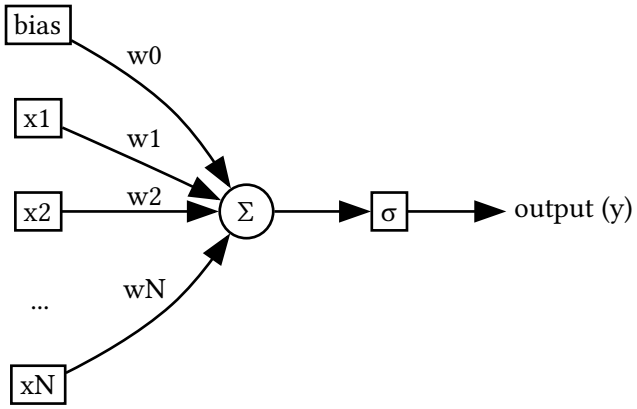
The boundary is computed so that:

- if $\boldsymbol{w} \cdot \boldsymbol{x}_i > 0$ then $i$ belongs to class A
- if $\boldsymbol{w} \cdot \boldsymbol{x}_i \leq 0$ then $i$ belongs to class B



A **perceptron** is a simple artificial neural network that classifies input into two categories; it is a single unit that takes inputs (one for each attributes plus an extra node), multiplies them by the weights producing an output (class A or B).




Attribute values activate the input layer, then they are multiplied by the weights $\boldsymbol{w}$ and summed at the output node. The output node passes the linear function $\boldsymbol{w} \cdot \boldsymbol{x}_i$ through the **threshold function**, $h£$ or $\sigma$:

$$\sigma(\boldsymbol{w}^T \boldsymbol{x}_i) = \begin{cases} 1 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x}_i > 0 \\ 0 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x}_i \leq 0 \end{cases}$$

The **error for Perceptron learning rule**, given a number of instances $m$, the target/observed value $y_i$, and the predicted value/perceptron output $\hat{y}_i$ is:

$$\text{squared error} = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

To train a Perceptron, we use the **error correction** method (a form of gradient descent).

```
w ← {0, ...} initialise w randomly or to zero
repeat until error is small
    for i ∈ {1, ..., m}:
        for j ∈ {1, ..., n}:
            w_j ← w_j + α(y_i − ŷ_i)x_{i,j}
```

Where $\alpha$ is the **learning rate**, i.e. constant specifying size of adjustments in each iteration.

## 2.4. Neural Networks

A **feed-forward neural network** has nodes connected by directed links, in only one direction (directed acyclic graph).

- Nodes are grouped into layers so that each node receives input only from nodes in the immediately preceding layer. (links are between consecutive layers $\ell$ and $\ell + 1$)
- It can either be **single-layer** (perceptrons; inputs connected to outputs) or **multi-layer** (**hidden layer(s)** between the input and output layers).
- Each link is associated with a weight $w_{i,j}$ and propagates $a_i$.
  Therefore $a_j$ is computed as a function of all incoming activations.

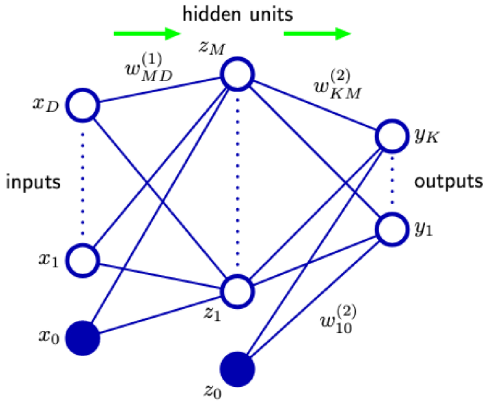



Figure 9: A feed-forward network

A **recurrent neural network** feeds outputs back into its own inputs.
Network activation levels form a dynamical system that may reach a steady state or exhibit oscillations and potentially chaotic behaviour. Often difficult to understand.

### 2.4.1. Activation Functions

The node $j$ in layer $\ell \geq 2$ computes the weighted sum $\sum_{i \in (\ell-1)}^{n} w_{i,j} a_i$ of all inputs.

The activation $a_j$ is obtained by applying an **activation function** $g(\cdot)$ to the weighted sum.

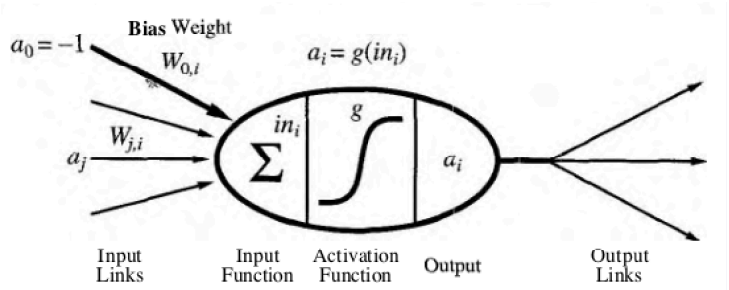


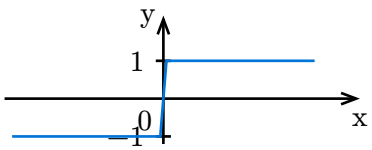

Figure 10: Neural Network Unit

Attractive properties of activation functions:

- **Non-linear**: for building models that generalise well
- **Differentiable**: for updating weights during learning
- **Monotonic**: for facilitating convergence

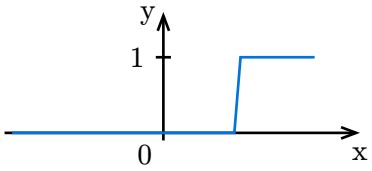
**Sign function**

$$g(s) = \begin{cases} 1, & s > 0 \\ -1, & s < 0 \end{cases}$$



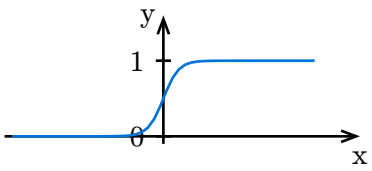

**Step function**

$$g(s) = \begin{cases} 1 & s \geq t \\ 0, & s < t \end{cases}$$



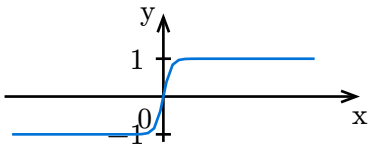

**Sigmoid function**

$$g(s) = \frac{1}{1 + e^{-s}}$$



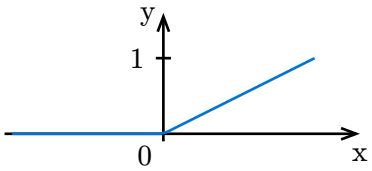

**Hyperbolic tangent function**

$$g(s) = \tanh(s)$$




**ReLU**

$$g(s) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases}$$



### 2.4.2. Feed-forward Neural Networks

**Initialisation**

Weights are initialised with a random small number.

**for** link $(i, j)$ **in** $\ell$-th layer:
  $w_{i,j} \leftarrow N\left(0, \frac{1}{n_\ell}\right)$ where $n_\ell$ is the no. of nodes

Initialisation critically affects the network performance:

- **Exploding gradient prbolem**: slightly increasing weights might exponentially increase gradients and the cost will oscillate around its minimum value.
- **Vanishing gradients problem**: slightly decreasing weights might exponentially decrease gradients and termination might occur before reaching the minimum error value.

**Forward-Pass**

Computing feed-forward NN outputs works by sequentially (on a layer-to-layer) basis, computing the outputs of nodes and carrying the results forwards.

**for** node $i$ **in** (input) layer $\ell = 1$:
  $a_i \leftarrow x_i$
**for** layer $\ell = 2..L$:
  **for** node $j$ **in** layer $\ell$:
    $\text{in}_j \leftarrow \sum_{i \in (\ell-1)} w_{i,j} a_i$
    $a_j \leftarrow g(\text{in}_j)$

**Training (with backpropagation)**

To train a feed-forward NN:

1. Initialise the network with some weights
2. Repeat until *termination criteria*:
   1. Perform a forward-pass to compute outputs
   2. Update network weights with *backpropagation* of the error

Backpropagation of the error occurs as such:

**for** node $j$ **in** (output) layer $\ell = L$:
  $\delta_j \leftarrow g'(\text{in}_j) \cdot (y_j - a_j)$
**for** layer $\ell = (L-1)..2$:
  **for** node $j$ **in** layer $\ell$:
    $\delta_i \leftarrow g'(\text{in}_i) \sum_{j \in (\ell+1)} w_{i,j} \delta_j$
**for** link $(i, j)$:
  $w_{i,j} \leftarrow w_{i,j} + \alpha \cdot a_j \cdot \delta_j$

A **termination criteria** is a sufficiently large number of iterations with a sufficiently small enough error on the training and validation set. Too few iterations/high error can lead to a poor model. Too many iterations/small error can lead to overfitting.

### 2.4.3. Evaluation

**Advantages of NNs**

- Perform a series of nonlinear transformations to the input.
- Highly parameterised means flexibility to model even small function irregularity.
- Models with few hidden layers can be proven to model any continous function.
- Flexibility can include danger of overfitting but modern techniques can avoid this.

**Disadvantages of NNs**

- Often unexplainable black boxes with limited ability to identify causal relationships.
- Require significant computational resources.
- Only works well with sufficiently large data sets.

# 3. Regression & Classification (Tree Models)

## 3.1. Classification Rules

A **rule** is an expression in **if-then** format where the if part is the pre-condition/antecedent and consists of a series of tests and the then part is the conclusion/consequent and assigns values to one or more attributes

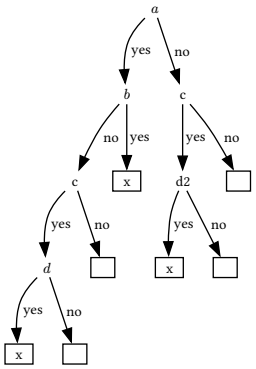
The pre-condition may contains multiple clauses in form of:
- conjunction: all tests must be true to fire the rule
- disjunction: at least one test must be true to fire
- general logic expressions: tests linked by logical operators, and/or

**Classification Rules** predict the class or label of an instance; can be derived from a decision tree where one rule is constructed for each leaf of the tree

The **replicated subtree problem** may occur where no matter which rule is chosen first, another is replicated in the tree; which means sometimes classification rules are significantly more compact

```
if a and b:
   return x

if c and d:
   return x
```




A set of rules *may fail* to classify an instance, which cannot happen with decision trees.

Rules can be un-/ordered:
- *ordered* set of rules: applied based on given order
  an individual rule out of the list may be incorrect
- *unordered* set of rules: each rule represents independent piece of knowledge
  different rules may lead to different classes for one instance

**Association Rules**
- predict an attribute of an instance
- similar to classification but can predict combinations of attributes too
- express different regularities in the data set
- many different association rules (even in small data sets)

Some interesting rules include high coverage (number of correctly predicted instances) and accuracy (% correctly predicted over all instances). We seek rules with coverage/accuracy above threshold.

**Learning Rules**
- by adding new rules and refining new rules while more instances are added in training set
- refinement may mean adding another conjunctive clause to a pre-condition
- rules may:
  - contain functions of attribute values, e.g. area(rectangle)
  - compare attribute values/functions of them, e.g. area(rectangle) > width(rectangle)
  - recursively concern different data set parts, e.g. tallerThan(rectangle, triangle)

A **rudimentary rule** is a rule where the condition tests only one attribute

**Algorithm 1R** is an approach to constructing rudimentary rules:
- For every attribute value, compute label that occurs more often (in training data).
- Return rudimentary rules with a reasonably low error rate, i.e. below a predefined threshold.

```
rules ← {}
begin by determining all reasonable rules:
for attribute j ∈ attributes:
    for value v ∈ values(j):
        build a rule based on most frequent class for this attribute value:
        for class c ∈ classes:
            | count(v, c) ← number of (v, c) occurences
        rules(v) ← arg max_c{count(v, c)}
then calculate rule error rates and choose rules with smallest error rates:
for r ∈ rules:
    error rate ← wrong predictions / all instances
    if error rate > threshold:
        | ~~rules(v)~~ (remove the rule)
```

Algorithm 1R has been shown to perform surprisingly well in some data sets, with performance comparable to the one of state-of-the-art learning schemes.

**Example: Building classifier from Weather Data (Table 2)**

Enumerate the possible attribute (for simplicity, only consider outlook) and class values:
- $\text{values}(j) = \{\text{sunny}, \text{overcast}, \text{rainy}\}$
- Predicted class **play**; $\text{classes} = \{\text{yes}, \text{no}\}$

Count no. of entries of combinations with each outlook and each class.
This would produce:

| | Attribute | Rules | Errors | Total Errors |
|---|---|---|---|---|
| 1 | outlook | sunny → no | 2/5 | 4/14 |
| | | overcast → yes | 0/4 | |
| | | rainy → yes | 2/5 | |
| 2 | temperature | hot → no* | 2/4 | 5/14 |
| | | mild → yes | 2/6 | |
| | | cool → yes | 1/4 | |
| 3 | humidity | high → no | 3/7 | 4/14 |
| | | normal → yes | 1/7 | |
| 4 | windy | false → yes | 2/8 | 5/14 |
| | | true → no* | 3/6 | |

### 3.1.1. Using numeric data

Rudimentary rules may accommodate numeric attributes, by converting them into nominal ones, using **discretization**. We create a sequence of sorted values with corresponding class and attempt to define breakpoints to produce different categories.

**Example of discretization**

Suppose we have numeric data, like Table 3, we could discretise it:

Discretization:

```
64   65  68   69   70   71  72  72   75   75   80  81   83   85
yes  no  yes  yes  yes  no  no  yes  yes  yes  no  yes  yes  no
```

Attempt to place breakpoints:

```
yes | no | yes yes yes | no no | yes yes yes | no | yes yes | no
```

To fix the issue with the temperature value 72, we may use the majority class in each category.

Approach for constructing rudimentary rules with numeric values:

- create a list of <attribute, class> parts
- discretize and sort values
- partition sorted list into groups with breakpoints
- choose class of each group with majority voting

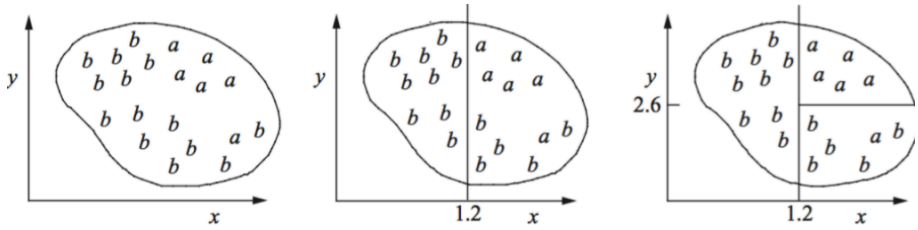
Because too many groups may result in *overfitting*, we may:

- specify a minimum number of instances per group,
- merge adjacent partitions with the same majority class

Missing values can be handled using attribute value *missing*.

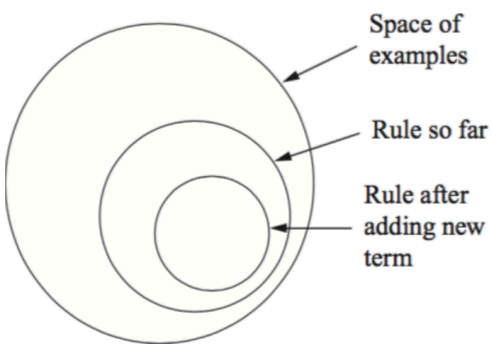
### 3.1.2. Covering Approach for Classification Rules

**Covering Approach for Classification Rules** Iteratively take each class and cover all instances in it.




Define a rule for a class and then test how much of the training set is *covered* by the rule.

- Start with an empty rule for predicting a desired class.
- Iteratively add new terms, i.e. tests, that better separate from the other classes.

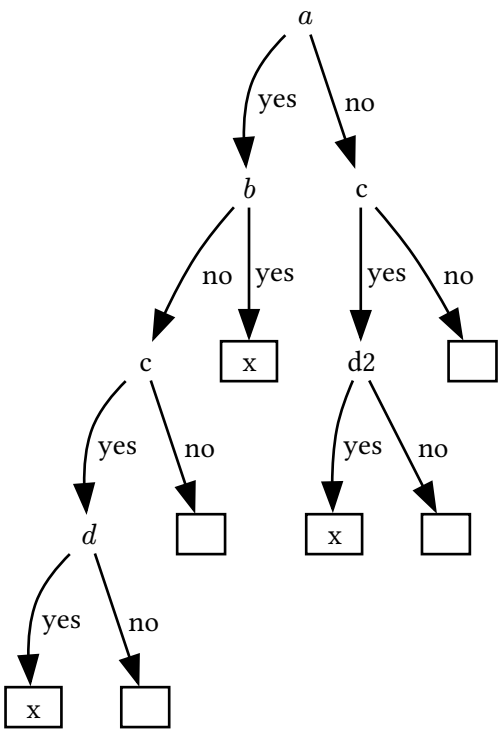


A rule may cover only a subset of instances of the class that it predicts. A rule may cover instances from multiple classes, apart from the one that it predicts. Computing rules with high accuracy:

$$\text{accuracy} = \frac{\text{number of correct predicted instances}}{\text{number of all instances covered}}$$

## 3.2. Decision Trees

A **decision tree** is a tree of **branches** and **nodes** where each branch may involve a single or multiple attributes; we examine the value of an attribute and branch based on (in-)equality




**Advantages of decision trees**

- Can handle mixed data types
- Can deal with missing values

**Disadvantages of decision trees**

- Decision boundaries are typically parallel to input axes
- Overfitting is potentially a significant problem

The **missing value problem** is where it is unclear which branch should be considered if an attribute value is missing; there are several solutions which each can propagate errors with lots of missing values: ignore all instances with missing values, each attribute may get value *missing*, set most popular choice for each missing attribute value, make probailistic weighted choice for each missing attribute value (based on other instances)

A **functional tree** is computes a function of multiple attributes in each node; branches based on value returned by the function

A **regression tree** is predicts numeric values; each node branches on the value of an attribute or on the value of a function of the attributes; a leaf specifies a predicted value for corresponding instances

A **model tree** is similar to a regression tree but instead we use a **regression equation** to predict the numeric output value in each leaf; the equation predicts a numeric quantity as a function of the attributes

### 3.2.1. Divide-and-Conquer Method for Constructing Decision Trees

**Divide-and-Conquer**

1. Select an attribute at the root
2. Make one split for each possible attribute value (assume finite set of possible values)
3. Repeat (1) for each child

We **recursively** break down the problem into two or more smaller subproblems, until these become simple enough. Then **combine** subproblem solutions to solve the original problem.

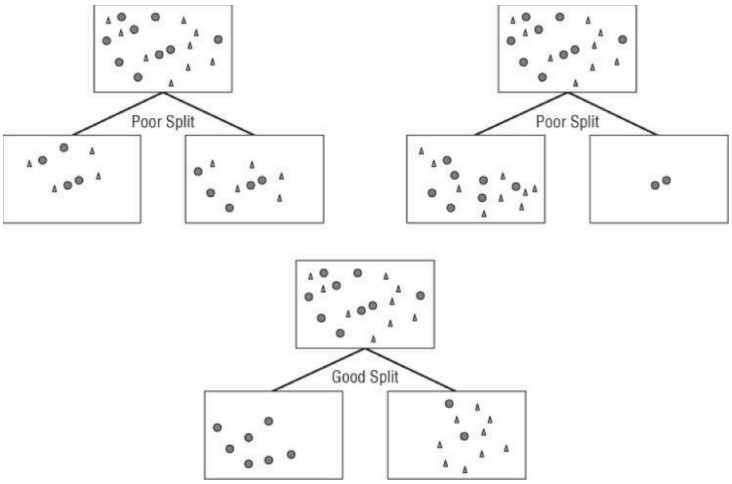



Figure 14: A good split increases **purity** for all children

A **split measure** is the purity of the resulting children nodes; high purity means members of a single class predominate; low purity means the distribution of classes in the children is similar to the one in the parent node.

Good splits should achieve high purity and generate nodes of similar size (e.g. avoid small nodes).

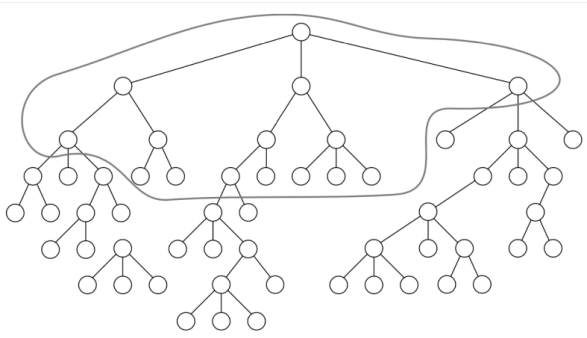
**Termination Criteria for Splitting**

- The tree depth reaches a limit.
- The number of instances in a node is sufficiently large.
- There are no splits left that may increase the node's purity.

**Potential Overfitting**

A decision tree captures general patterns in nodes high in the tree and patterns which are more specific to the training set deeper in the tree. The latter splits may be the cause of overfitting.
We could deal with this by:

- setting a sufficiently large min. leaf size
- continue growing tree as long as splits are significant
- use **pruning** to eliminate insignificant splits



### 3.2.2. State-of-the-Art Purity Measures

Several purity measures (Gini, Information Gain/Entropy Reduction, $x^2$ Test) may result in different splits. Resulting models tend to behave similarly because all measures attempt to capture same ideas.

#### 3.2.2.1. Gini Impurity

The **Gini Coefficient / Gini Impurity Measure** is the sum of squared proportions of the classes, given the probability $\Pr(k)$ an instance belongs to a class $k$, where $C$ is the number of classes:

$$G = \sum_{k=1}^{C} \Pr(k)^2$$

Inspired by population diversity in biology.

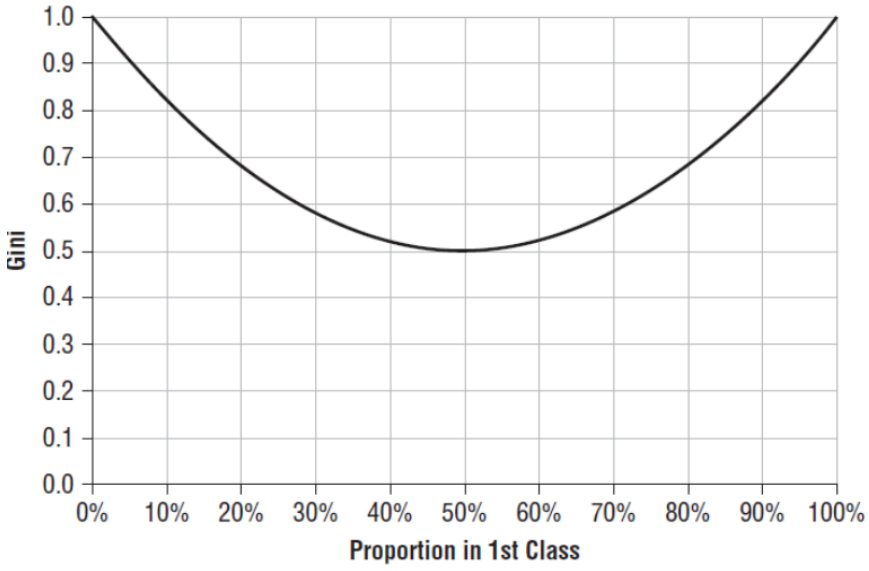
> **Note**
>
> The Gini measure is more widely defined as
>
> $$G = 1 - \sum_{k=1}^{C} \Pr(k)^2$$

The Gini Coefficient gives the probability that two random items (sampling with replacement) from the population belong to the same class:

- When $G = 0.5$: two represented equally (impurity)
- When $G = 1$: all items in one class (pure)

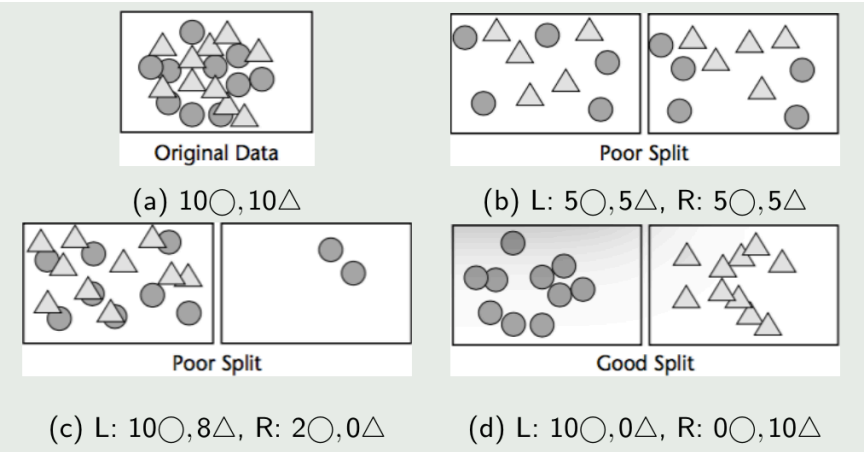
We aim to select the split for which the average Gini score of the children is maximal.

> **Note**
>
> For binary classification, the Gini measure varies from 0.5, when there is an equal number of instances to each class, to 1, when all instances are in the same class.



**Example using Triangles & Circles Data Set**

Suppose the following splits:




(b) $G(L) = \Pr(\bigcirc)^2 + \Pr(\triangle)^2 = (0.5)^2 + (0.5)^2 = 0.5$
(c) $G(L) = \Pr(\bigcirc)^2 + \Pr(\triangle)^2 = (0.56)^2 + (0.44)^2 = 0.5072$
(d) $G(L) = \Pr(\bigcirc)^2 + \Pr(\triangle)^2 = (1.0)^2 + (0.0)^2 = 1.0$

#### 3.2.2.2. Entropy

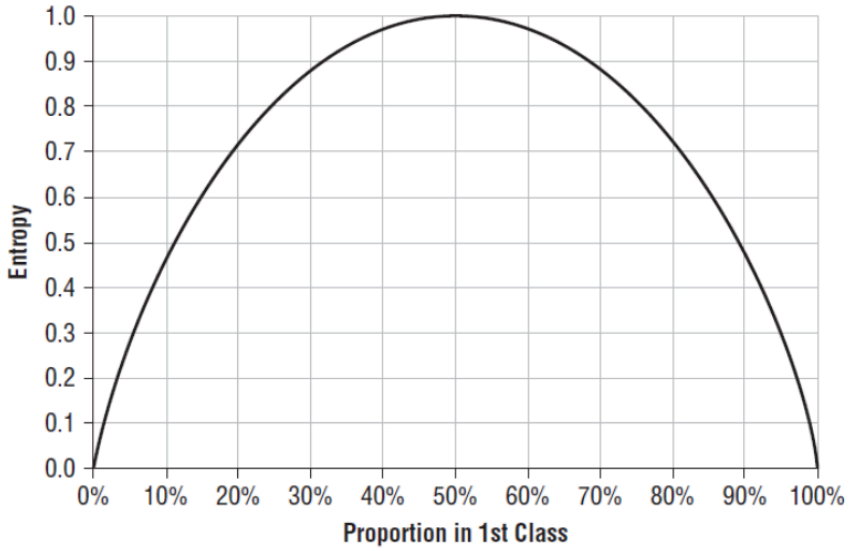
**Entropy** is a measure of disorder of a system, quantifying the uncertainty associated with a random variable (many low prob. events have high entropy, few high prob. events have lower entropy). Computes the amount of information in number of bits. The higher the information aqcuisition, the higher the entropy. The more certain an event is, the less information it contains.

Given the probability $\Pr(k)$ an instance belongs to a class $k$, where $C$ is the number of classes:

$$E(\text{ntropy}) = (-1) \cdot \left[ \sum_{k=1}^{C} \Pr(k) \cdot \log_2(\Pr(k)) \right]$$

**The Gini measure multiplies each probability by itself, while Entropy multiplies each probability by the logarithm of itself.** The proportion of records belonging to a particular class multiplied by the base two logarithm of that proportion. Multiplication by –1 ensures positive values, since the logarithm of a probability is negative.

> **Note**
>
> For binary classification, the entropy varies from 0, when there is a pure population, to 1, when there is an equal number to each class.



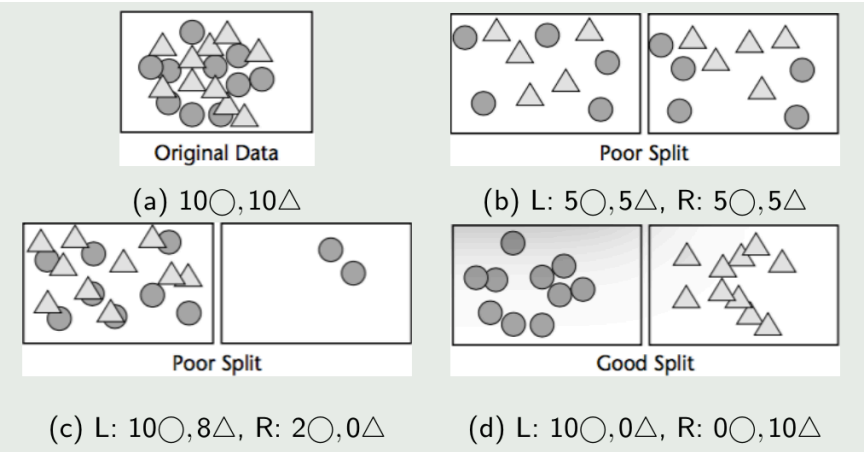

If a leaf is entirely pure, then the entropy is low. If a leaf is not pure, then the entropy is high.
So, determine splits maximising **information gain** / **entropy reduction**:

$$\text{IG} = E(v) - [f_1 E(v_1) + ... + f_\ell E(v_\ell)]$$

Where $v$ is the parent node, $v_1, ..., v_\ell$ are child nodes, and $f_k$ is the proportion of instances in child node $v_k$ over all of the instances.

## Example using Triangles & Circles Data Set

Suppose the following splits:




(b) $E(L) = (-1) \cdot (0.5 \cdot \log_2(0.5) + \Pr(0.5) \cdot \log_2(0.5)) = 10$
(c) $E(L) = (-1) \cdot (0.56 \cdot \log_2(0.56) + \Pr(0.44) \cdot \log_2(0.44)) = 0.9896$
(d) $E(L) = (-1) \cdot (1.0 - \varepsilon) \cdot \log_2(1.0 - \varepsilon) + (\varepsilon) \cdot \log_2(\varepsilon) = 0.0$

#### 3.2.2.3. Chi-Square ($\mathcal{X}^2$)

The **Chi-Square**, $x^2$, is, given the observed fraction of class $k$ instances in child node, $O_k$, and the expected fraction of class $k$ instances from parent node, given by:

$$x^2 = \sum_{k=1}^{C} \frac{(O_k - E_k)^2}{E_k}$$

> **Note**
>
> Can be calculated using a **contigency table** for the split.

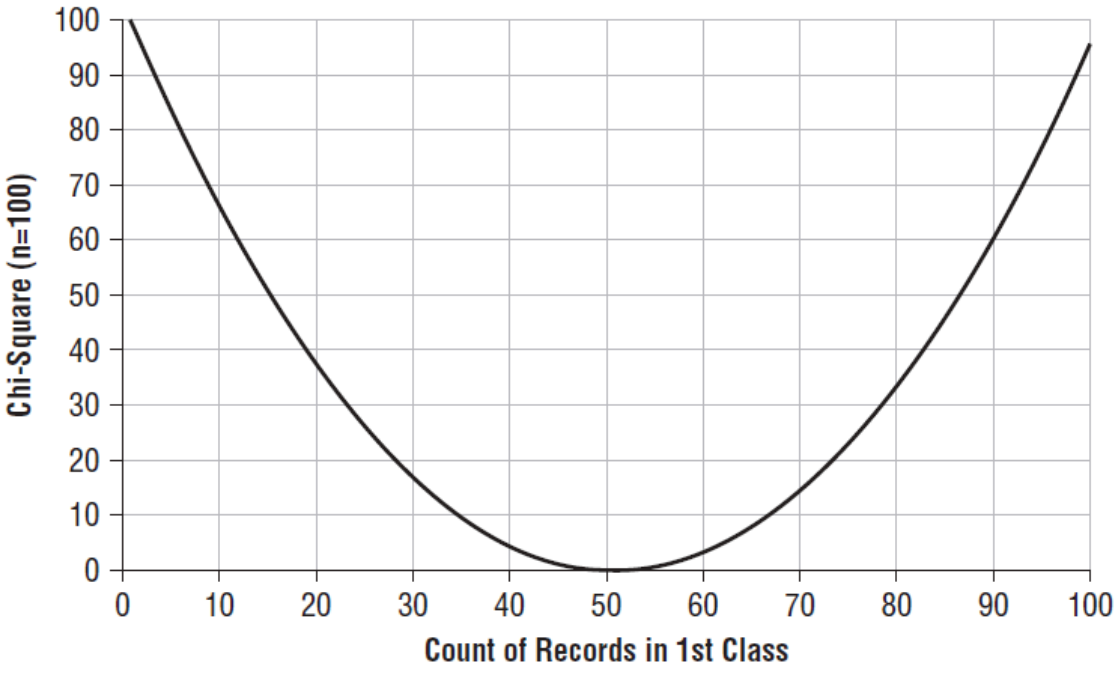
| | **Class 1** | **Class 2** |
|---|---|---|
| **Left child** | $L_1$ | $L_2$ |
| **Right child** | $R_1$ | $R_2$ |

For the left child:

- Observed freq.: $O_1 = L_1$
- Expected freq.: $E_1 = \frac{L_1+R_1}{L_1+L_2+R_1+R_2}(L_1 + L_2)$

Evaluate split with the $\mathcal{X}^2$ average over all children.

> **Note**
>
> Suppose a data set with 100 instances.
> For binary classification, when $\mathcal{X}^2$ is zero, then the distribution of the children is similar to their parent's. Not restricted to $[0, 1]$ like Gini and entropy are.



### 3.2.3. Decision Tree Learning Algorithms

Overview of typical algorithms:

| | Input | Split selection | Pruning |
|---|---|---|---|
| **CART** | Numerical and categorical | Gini index | Adjusted error rate |
| **ID3** | Categorical only | Information Gain | No |
| **C5.0** | Numerical and categorical | Information Gain | Error rate computation with statistical confidence intervals |

#### 3.2.3.1. Classification And Regression Trees (CART)

In **Classification and Regression Trees (CART)**, we grow binary trees and continue to grow them as long as we can find new splits that increase purity. Once completed, we prune:

- Replace any subtree whose leaves make the same classification with their common parent.
- Prune splits with the least predictive power per leaf, which can be identified by computing the **adjusted error rate**, a measure of a node's error rate penalized by the number of leafs in the tree.

$$\mathrm{AE}(R) = E(R) + \alpha \cdot \mathrm{leafs}(R)$$

Good trade-off between error rate and no. of leafs.

#### 3.2.3.2. Iterative Dichotomizer (ID3)

**Iterative Dichotomizer (ID3)** works as follows:

- Randomly choose a window, i.e. a subset of the training set.
- Build a decision tree using the window.
- Apply the decision tree to all the other instances in the training set.
- If the error rate is sufficiently good, terminate.
- Otherwise, add misclassified instances in the window and repeat.

A suitable measure for choosing the next split is the expected amount of information.

- Can be imagined as answering a question, e.g. whether a coin will land heads, measured in **number of bits**: 1 bit of information to answer a yes/no question to someone without knowledge about answer; a four-sided dice has 2 bits of entropy.
- Depends on prior knowledge, i.e. the less we know, the more information is provided.
- If there are $\ell$ possible answers $v_1, ..., v_\ell$ with probability $\Pr(v_k)$ then amount of information is:

$$E(v_1, ..., v_\ell) = -\sum_{k=1}^{\ell} \sum_{k=1}^{\ell} \Pr(v_k) \log_2 \Pr(v_k)$$

> **Note**
>
> Suppose a binary classifier, $p$ number of possible instances (class $P$) and $n$ negative (class $N$). Any correct decision tree for $l$ will create partitions that are proportional to representations of instances in $I$: $\Pr(i \in P) = \frac{p}{p+n}$ and $\Pr(i \in N) = \frac{n}{p+n}$.
>
> Estimated information in a correct answer is:
>
> $$E(p, n) = -\left(\frac{p}{p+n}\right) \log_2 \left(\frac{p}{p+n}\right) - \left(\frac{n}{p+n}\right) \log_2 \left(\frac{n}{p+n}\right)$$

Suppose that attribute $x_j$ with values $\{v_{j,1}, ..., v_{j,\ell}\}$ is chosen at root of $R$ of a decision tree.
The split will create $\ell$ children nodes $R_1, ..., R_\ell$ and partition the training set into $\ell$ subsets.
Node $R_k$ contains all instances with $x_j = v_{j,k}$, for each $k = 1, ..., \ell$.
Suppose that $R_k$ contains $p_k$ instances of class $P$ and $n_k$ instances of class $N$.
If we go along this branch, we need $E(p_k, n_k)$ bits of information to answer the question.

A randomly chosen instance $i$ for training set has $x_{i,j} = v_{j,\ell}$ with probability $\frac{p_k+n_k}{p+n}$.
Hence, after testing attribute $j$, remaining expected entropy to classify example is:

$$\sum_{k=1}^{\ell} \frac{p_k + n_k}{p + n} E(p_k, n_k)$$

The **information gain** is:

$$\text{IG} = E(p, n) - \sum_{k=1}^{\ell} \frac{p_k + n_k}{p + n} E(p_k, n_k)$$

#### 3.2.3.3. C5.0

**C5.0** is similar to CART, where we first grow an overfitted tree. It works as follows:

- Make multi-way splits on categorical variables.
- Use information gain for splitting.
- Given the error rate and size of a node, the error rate on unseen data can be predicted with an upper bound of the statistical confidence interval where the true error rate lies.
- When the error estimate at a node is less than the error estimate of the children, then the children can be pruned.

**Statistical Confidence Interval Upper Bound for Error Rate** Given the misclassification rate of tree $T$ on data set $S$, $\varepsilon(T, S)$, the inverse of the standard normal cumulative distribution, $Z$, and the desired significance level, $\alpha$, statistical confidence interval upper bound for error rate is given by:

$$\varepsilon_{\text{UB}(T,S)} = \varepsilon(T, S) + Z_\alpha \cdot \sqrt{\frac{\varepsilon(T, S) \cdot (1 - \varepsilon(T, S))}{|S|}}$$

#### 3.2.3.4. Ensemble Methods

Other approaches based on ensemble methods:

- **Random Forest**: randomly select subset of attributes to learn a tree on; repeat multiple times; predict new instances by aggregating predictions of individual trees (e.g. by voting)
- **Gradient Boosting Tree**: a weak learner is used to construct a strong learner; at each step, a new tree is added to improve the model containing only the previous trees

## 3.3. Cross-validation and Bootstrapping

Performance on the training set is usually not a good indicator of prediction quality for new instances. To assess the performance of a classifier on new data, we need a test set that is not used in any way for training the classifier.

Useful for estimating the error rate of predictive models on unseen instances.

**Holdout Procedure**: setting data aside for testing

The **Error Rate of Classification Models** is the proportion of errors over all instances:

$$\text{error rate} = \frac{\text{incorrect}}{\text{correct} + \text{incorrect}}$$

The **Resubstitution Error** is the error rate on training data; calculated by resubstituting the training instances into the classifier that was built from them; even though not reliable predictor of the true error rate on new data, useful to know

We may decompose data set into three independent sets:
- Training set: used to learn classification models
- Validation set: used to select best model type/hyperparameters
- Test set: used to evaluate classification models

With a massive data set, we may select one large sample for training and another, independent large sample for testing. In the case of limited size data sets, there is a trade-off between selecting a sufficiently large:
- training set to derive a good classifier, and
- a disjoint test set to obtain a good error estimate.
  We may use *cross-validation* or *bootstrapping*.

**Stratification** or **stratified holdout**: the process of randomly sampling to create the training and test sets so that they contain proportional representations of each class

**Repeated holdout**: process of repeating training and testing several times with different random samples; mitigates bias of particular samples chosen for holdout

### 3.3.1. Cross-validation

**Cross Validation** is the process of splitting the data into folds to repeatedly train and test models using different combinations of folds

***N*-Fold Cross Validation** is the process of splitting the data into approximately $N$ equal folds to repeatedly train and test models using different combinations of folds; perform $N$ repetitions where one fold is used for testing and remaining are used for training; compute error rate $N$ times after each repetition and average results to yield overall error rate

A common strategy is **10-times 10-fold cross validation** where we repeat the process 10 times with random sampling for splitting the data set. There is experience and theoretical evidence justifying this strategy and it is often used in practice.

**Leave-One-Out Cross-Validation**: use $N$-fold cross-validation where $N$ is the total number of instances in the data set; test set is exactly 1 and a maximal amount of data is used for training.
No random sampling so no chance of *sampling bias*.
Error rate may be artificially high/low due to lack of stratification.

Cross-validation compared to Bootstrapping:
- Cross-validation uses sampling without replacement; every selected instance is removed from the data set and cannot be used again; data set is shuffled
- Bootstrapping uses sampling with replacement; every selected instance remains in data set

### 3.3.2. Bootstrapping

**Bootstrapping** or **0.632-Bootstrap** in a data set with $n$ instances works as follows:
1. Construct the training set by performing $n$ trials using sampling with replacement.
2. In each trial, add every chosen instance in the training set.
3. Once the sampling process is over, use the non-selected instances as the test set.

Each instance has a $\frac{1}{n}$ probability to be chosen in one trial, and a $1 - \frac{1}{n}$ to not be chosen.
Probability that an instance is not added in the training set is:

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

Training set size is roughly $63.2\%$ of all instances and test set size is roughly $36.8\%$ of all instances.

0.632-Bootstrap has significantly smaller training set than 10-fold cross-validation which uses 90% of the instances. Therefore, will give pessimistic estimates of the true error rate.

To compensate for this, we may combine the *resubstitution error* with the *test-set error* to obtain better estimates of the true error rate:

$$\text{error rate} = 0.632 \cdot \text{test error} + 0.368 \cdot \text{training error}$$

Repeat several times with different samples and average the results to obtain the overall error rate.

NB. Bootstrap error rates can still be artificially high or low

## 3.4. Classification Model Evaluation

### 3.4.1. Confusion Matrix

In a binary classification problem with two classes (yes/no):

- **True Positives, TP**: number of instances correctly predicted as yes
- **True Negatives, TN**: number of instances correctly predicted as no
- **False Positives, FP**: number of instances incorrectly predicted as yes
- **False Negatives, FN**: number of instances incorrectly predicted as no

Confusion matrix is therefore:

| | | **Predicted Class** | |
|---|---|---|---|
| | | **Yes** | **No** |
| **Actual Class** | **Yes** | True positive | False negative |
| | **No** | False positive | True negative |

We can compute the:

- **Success Rate**: fraction of correctly predicted instances over all instances

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

- **Error Rate**: $1 - \text{success rate}$
- **True Positive Rate**: percentage of yes instances correctly predicted

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

- **False Positive Rate**: percentage of no instances predicted incorrectly yes

$$\frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

In multi-class prediction problem, the confusion matrix is a 2-dimensional matrix with a row and a column for each class. The correctly predicted instances lie in the diagonal.

| | | **Predicted Class** | | | |
|---|---|---|---|---|---|
| | | ***a*** | ***b*** | ***c*** | ***Total*** |
| **Actual Class** | ***a*** | 88 | 10 | 2 | 100 |
| | ***b*** | 14 | 40 | 6 | 60 |
| | ***c*** | 18 | 10 | 12 | 40 |
| | ***Total*** | 120 | 60 | 20 | |

### 3.4.2. Cohen's Kappa

**Cohen's Kappa Coefficient/Kappa Statistic** compares the agreement between two classifiers, given a proportion of agreement $\Pr(a)$ and proportion of expected agreement $\Pr(e)$:

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)}$$

We can build an **agreement matrix** where we show the no. of instances where both $C$ and $C'$ classify as $\frac{A}{B}$, and cases where they don't:

| | | **C'** | |
|---|---|---|---|
| | | **Class A** | **Class B** |
| **C** | **Class A** | a,a | a,b |
| | **Class B** | b,a | b,b |

And hence, we can find:

- **Number of instances**: $n = a, a + a, b + b, a + b, b$
- **Proportion of agreement**:

$$\Pr(a) = \frac{a, a + b, b}{n}$$

- **Proportion of expected agreement**:

$$\Pr(e) = \left(\frac{a, a + b, a}{n}\right)\left(\frac{a, a + a, b}{n}\right) + \left(\frac{b, b + a, b}{n}\right)\left(\frac{b, b + b, a}{n}\right)$$

### 3.4.3. Lift

**Lift** evaluates classifiers predicting class membership probabilities by comparing results obtained with and without the classifier and considering subsets of instances.

**Marketing Example**

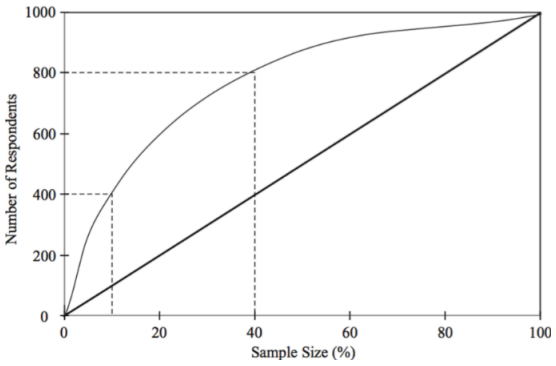
For a promotional offer to 1,000,000 individuals, we expect a 0.1% response rate.

- For a subset of 100,000 among these individuals which are the most likely to respond according to a classifier, the predicted response rate is 0.4%.
- The factor 4 increase in the response rate is the lift yielded.

**Computation of Lift**

- Sort instances in descending order of predicted probabilities 'yes', which have been computed by the classifier.
- Given a sample size, select the corresponding number of most likely 'yes' instances according to our model.
- Calculate the lif factor by counting the number of true positive instances in the sample and divide it by the sample size.

The **lift curve** is the number of successes in the equal size sample with the top ranked instances. The diagonal line is the expected number of 'yes' instances in a randomly selected sample.




The greater the area between the two curves, the better.

**Example**

**Table 5.6** Data for a Lift Chart

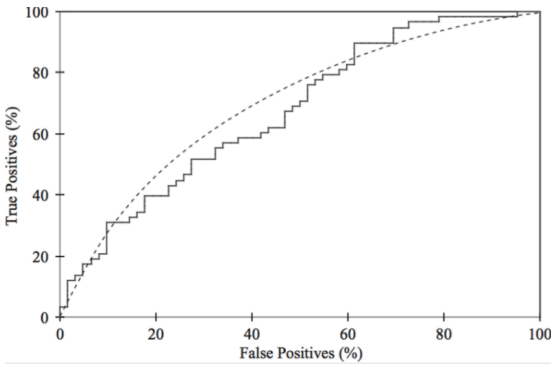
| Rank | Predicted | Actual Class |
|---|---|---|
| 1 | 0.95 | *yes* |
| 2 | 0.93 | *yes* |
| 3 | 0.93 | *no* |
| 4 | 0.88 | *yes* |
| 5 | 0.86 | *yes* |
| 6 | 0.85 | *yes* |
| 7 | 0.82 | *yes* |
| 8 | 0.80 | *yes* |
| 9 | 0.80 | *no* |
| 10 | 0.79 | *yes* |
| 11 | 0.77 | *no* |
| 12 | 0.76 | *yes* |
| 13 | 0.73 | *yes* |
| 14 | 0.65 | *no* |
| 15 | 0.63 | *yes* |
| 16 | 0.58 | *no* |
| 17 | 0.56 | *yes* |
| 18 | 0.49 | *no* |
| 19 | 0.48 | *yes* |
| ... | ... | ... |

- Number of instances: 150
- Classes: yes, no
- Number of 'yes' instances: 50
- In 10 randomly selected instances, 33.33% are 'yes'
- In top 10 ranked instances, 80% will be 'yes'

### 3.4.4. ROC

The **Receiver Operating Characteristic** (ROC) is used in signal detection to relate the *true positive rate* (hit rate) with the *false positive rate* (false alarm rate) over a noisy channel.

Similarly to lift, ROC is computed based on the highest-ranked instances.




For each number of no's, take enough highest ranked instances to include that number of no's and count the number of yes's that they contain.

Often, the ROC and lift curves look very similar.

We may compare two classification models using the **Area Under the Curve (AUC) of ROC**. Higher AUC is better. If AUC is 100% of the available area, then the model is perfect.

# 4. Clustering

. **Cluster Analysis or Clustering** is the process of organizing the instances clusters, i.e. dividing a data set into groups, so that:

- similar instances belong to the same cluster
- dissimilar instances appear in different clusters

We use clustering to discover natural, potentially unknown, subclasses of objects.

## 4.1. Clustering Techniques

### 4.1.1. K-means

The basic **K-Means Algorithm** works as follows:

1. Select $K$ points $c_1, ..., c_K$ as centroids
2. Assign each point $x_i$ to its closest centroid $c_K$ and form $K$ clusters $C_1, ..., C_K$
3. Recompute a new centroid $c_K = \frac{1}{m_k} \sum_{i \in C_k} x_i$ for each cluster $C_k$, where $k = 1, ..., K$
4. For each point $x_i$, find the closest centroid
5. Repeat steps 3—4 until centroids do not change (we have **converged**)

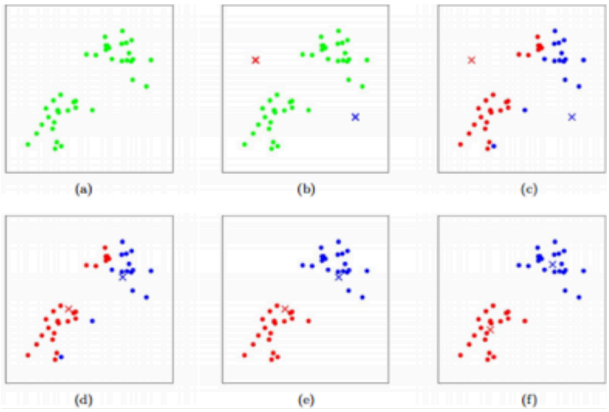



Figure 23: Illustration of K-means steps

The K initial centroids $c_1, ..., c_K$ can be either chosen randomly, or in more sophisticated ways.

To calculate the closest centroid $c_k$ for each point $x_i$, we need a **distance metric**:

- For numerical data, we might use **Euclidean distance**:

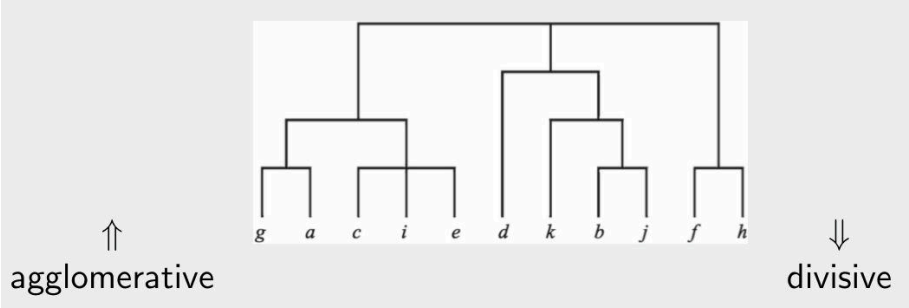
$$d(x_i, c_k) = \sqrt{\sum_{j=1}^{n} (x_{i,j} - c_{k,j})^2}$$

- For nominal data, we may compute centroids based on the mode, i.e. most popular value, and calculate distances using the *hamming distance*.

#### 4.1.2. Hierarchical

**Agglomerative Hierarchical Clustering** is iteratively merges clusters from individual points as its own cluster

**Divisive Hierarchical Clustering** is iteratively splits clusters from one cluster made of all items




**Agglomerative method**

1. Start with singleton clusters (each with one data point)
2. Repeat until one global cluster with all data points remains:
   1. Find a pair $(C_k, C_{k'})$ of clusters which are close
   2. Merge these clusters and replace with the newly created cluster
3. A dendrogram is produced.

This process requires a **between cluster distance metric**, $d(C_k, C_{k'})$.
Seen in distance metrics section later on.

**Divisive method**

1. Start with one global cluster (containing all data points)
2. Iteratively split clusters until singleton clusters remain
   There may be several approaches used to decide how to split in each step and how to partition instances between clusters, e.g.
   - split a cluster with high within cluster distance
   - construct two sub-clusters with high between cluster distance.

### 4.1.3. Density-based

The density of a data point $x$ is the number of data points within a specified radius *Eps*, including $x$.
This is a 'centered-based definition'. Other definitions also exist, but not as many for similarity.

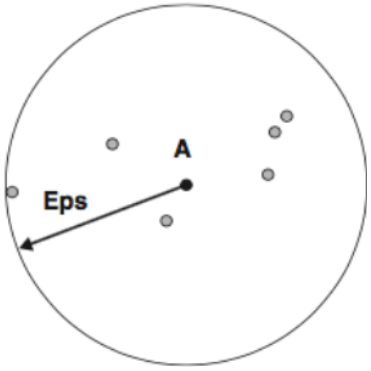



Figure 25: Labelled point $A$ has density 7

Different points may be categorised differently:

- **Core point**: has density above threshold *MinPts*
- **Border point**: not a core point, but lies in neighbourhood of core point
- **Noise point**: any other point

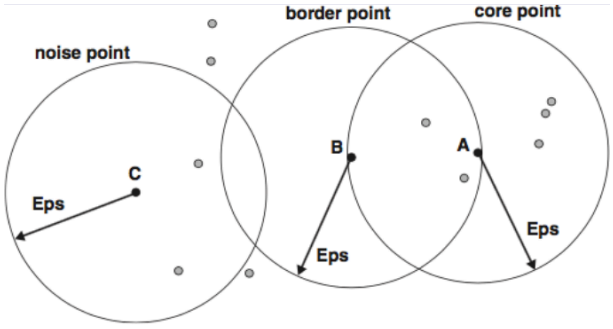



Figure 26: Data point categorisation

**DBSCAN Algorithm** Categorise all data points (core, border, noise) then, ignoring noise points, add an edge between every two core points of distance $\leq$ Eps. Create a cluster for each group of connected core points, assign each border point to the cluster of its core point.
Classic density-based clustering algorithm, identifies high density regions separated by low density regions between them.

Determining parameters *Eps* and *MinPts* is a key issue.

Selection depends on behaviour of the distance $k$-dist between the data points and their $k$-th nearest neighbours. For points that appear in some DBSCAN cluster, $k$-dist is small if $k$ is not larger than the cluster size. On the other hand, $k$-dist is large for points than do not appear in some cluster.

If we compute k-dist for each data point, sort these values in increasing order and plot the sorted values, there is a sharp change. Using this $k$-dist as *Eps* and $k$ as *MinPts*, while result points with $k$-dist less than *Eps* labelled as core points, while all other points will be labelled as border or noise points.

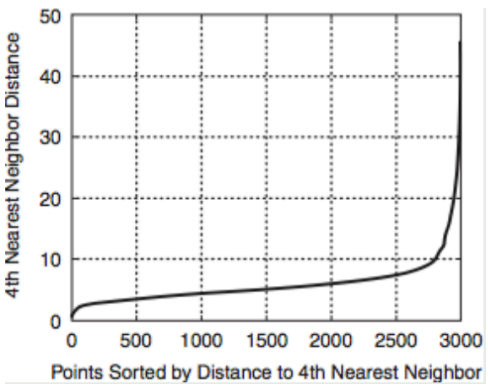


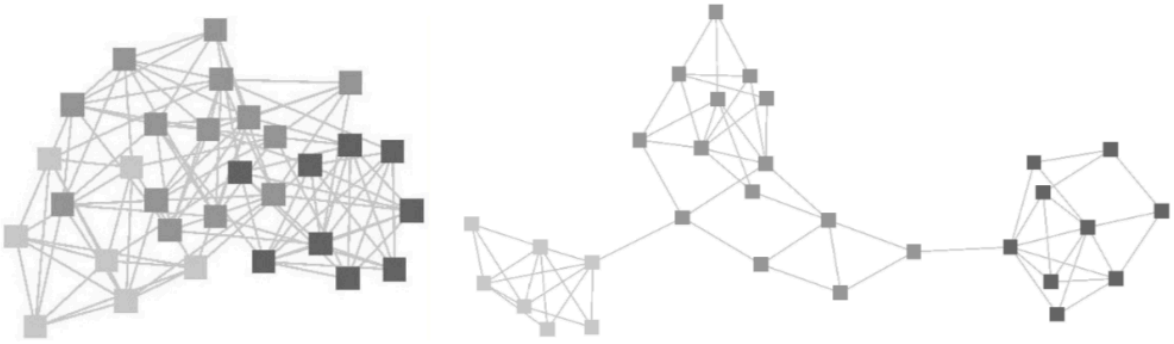

Figure 27: when $k = 4$, sharp occurs at $k$-dist = 4
*MinPts* = 4, *Eps* = 10

#### 4.1.4. Graph-based

A data set can be viewed as graph where nodes correspond to instances and edges between nodes have lengths equal to the corresponding distances. There are many clustering approaches here.

##### 4.1.4.1. Sparsification

In **sparsification**, retain graph nodes and filter out edges so that the cluster structure is retained or even enhanced in the sparsified graph, e.g. keep only edges between near neighbours.

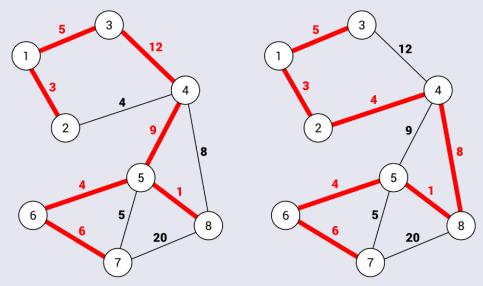


(a) Original graph (b) Sparsified graph (Sp. ratio=30%)

##### 4.1.4.2. Using similarity measures

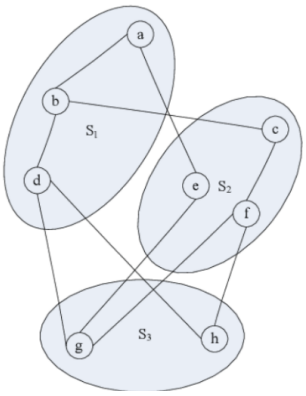
A **spanning tree** is a subset of edges connecting all vertices without a cycle.
A **minimum spanning tree** is a spanning tree that has minimal total edge length.

There exist very efficient algorithms for computing MSTs. An MST is in effect a sparsified graph. State-of-the-Art sparsification algorithms are based on sampling MSTs.




##### 4.1.4.3. Graph Partitioning

Standard clustering or minimum cut algorithms can be applied to the sparsified graph, e.g. using **minimum $k$-cut** to find a subset of edges whose removal partitions a graph into $k$-connected components. There exist efficient algorithms for min. $k$-cut.

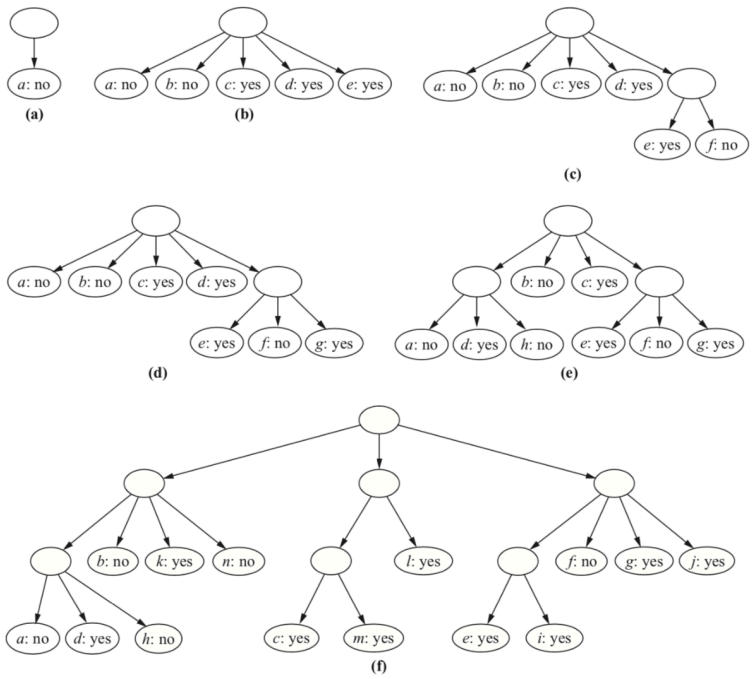


### 4.1.5. Incremental

In contrast to incremental, K-means iterates over entire data set until convergence; Agglomerative Hierarchical methods examine all current clusters whenever merging is performed.

**Incremental approaches** cluster instances one by one in online manner without requiring simultaneous access to all instances:

- Each stage, overall clustering forms a tree
- Root node represents the entire data set
- Leaves correspond to instances
- Nodes within the same parent form a cluster




**Incremental clustering steps**

1. Initialise a tree containing only a root node
2. With the addition of a new instance, the tree is updated:
   - either a leaf for the new instance is added,
   - or the tree is restructured by merging/splitting two nodes

Decisions (for a new instance, evaluating which node is best host) are taken based on the **category utility metric** (tree maintains category utilities obtained after merging/splitting):

$$\mathrm{CU} = \frac{1}{K}\sum_{k=1}^{K}\Pr(C_k)\left[\sum_{j=1}^{n}\sum_{\ell=1}^{r_j}\Big(\Pr\big(x_j = v_{j,\ell} \mid C_k\big)^2 - \Pr\big(x_j = v_{j,\ell}\big)^2\Big)\right]$$

Where $K$ is number of clusters, $n$ is number of attributes, $r_j$ is number of possible values for attribute $j$, $C_k$ is cluster $k$, $x_j$ is attribute $j$, $v_{j,\ell}$ is the $\ell$-th possible value for attribute $j$.

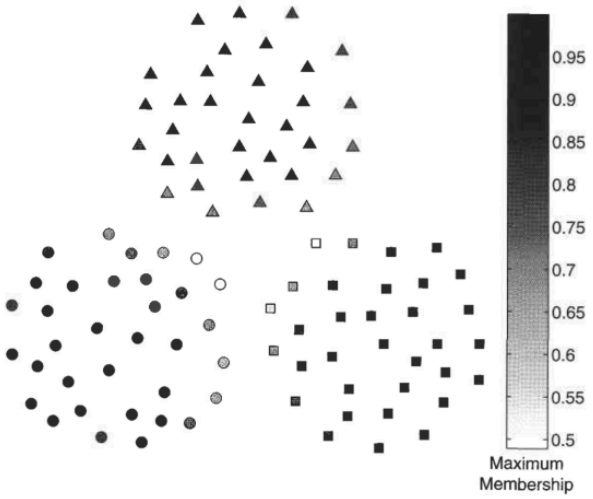
$\Pr\big(x_j = v_{j,\ell} \mid C_k\big)$ is the probability attribute $x_j$ has value $v_{j,\ell}$
$\Pr\big(x_j = v_{j,\ell} \mid C_k\big)$ is a similar probability, except we consider instances in cluster $C_k$ and not the entire data set.

For good clusters, the latter should be larger when summing squared probabilities; there is some advantage to predicting attribute values from the same cluster.

Division by $K$ assists in avoiding overfitting. Otherwise, singleton clusters would be the best choice.

#### 4.1.6. Fuzzy

A **fuzzy cluster** is one in which elements have varying degrees of membership.
Each instance $x_i$ is associated with a weight $w_{i,k} \in [0, 1]$ specifying the degree at which $x_i$ belongs to cluster $C_k$. All weights of an instance $x_i$ should sum to 1, i.e. $\sum_{k=1}^{K} w_{i,k} = 1$.
Each cluster $C_k$ contains at least one point with non-zero weight and cannot fully contain all points.




- Mixture models are useful for fuzzy clustering.
- Fuzzy variants of k-means can also be used.

## 4.2. Distance metrics and evaluation

**Distance metrics** measure similarity; they are, in some sense, the opposite of distance (greater similarity means elements are closer together).

We deal with three distance types:

- distance between values of two different instances with respect to an **individual attribute**

$$d(x_{i,j}, x_{i',j})$$

- distance between two different instances which **aggregates all attributes**

$$d(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$$

- distance between two different clusters which **aggregates all instances**

$$d(C_k, C_{k'})$$

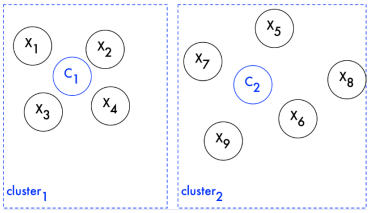
### Cluster Centroid for Numeric Data

Given the $k$-th cluster $C_k$, the $k$-th cluster centroid $c_k$, the $k$-th cluster size $m_k$, and the $j$-th attribute of $i$-th instance $x_{i,j}$:

$$c_k = \left( \frac{1}{m_k} \sum_{i \in C_k} x_{i,1}, \frac{1}{m_k} \sum_{i \in C_k} x_{i,2}, ..., \frac{1}{m_k} \sum_{i \in C_k} x_{i,n} \right)$$

### Cluster Medoid for Nomial Data

The most representative point for a group of points, i.e. most representative attribute values.




### Distance between Numeric Attribute Values

- Absolute difference/Manhattan distance: $|a - b|$
- Squared difference: $(a - b)^2$
- Euclidean distance: $\sqrt{(a - b)^2}$
- Normalised absolute difference obtained by dividing with max-min difference
- Absolute difference of standardised values which subtract the mean and divide by std-dev.

### Distance between Nominal Attribute Values

$$\text{hamming distance} = \begin{cases} 1, \text{if } a \neq b \\ 0, \text{otherwise} \end{cases}$$

### Cluster Diameter

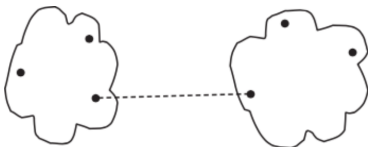
Largest distance $\text{diam}(C_k) = \max_{i,i' \in C_k} \{d(\boldsymbol{x}_i, \boldsymbol{x}_{i'})\}$ between any two points in a given cluster $C_k$.

#### 4.2.1. Cluster Linkage

**Single-Linkage**
The minimum distance

$$d(C_k, C_{k'}) = \min_{i \in C_k, i' \in C_{k'}} \{d(x_i, x_{i'})\}$$

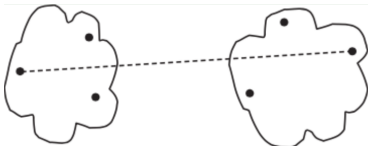
between any two points in the two clusters $C_k$ and $C_{k'}$



**Complete-Linkage**
The minimum distance

$$d(C_k, C_{k'}) = \max_{i \in C_k, i' \in C_{k'}} \{d(x_i, x_{i'})\}$$

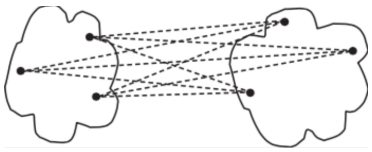
between any two points in the two clusters $C_k$ and $C_{k'}$



**Average-Linkage** (or Group Average)
The minimum distance

$$d(C_k, C_{k'}) = \frac{1}{m_k \cdot m_{k'}} \sum_{i \in C_k} \sum_{i' \in C_{k'}} \{d(x_i, x_{i'})\}$$

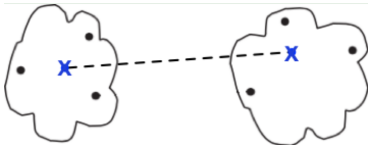
among all pairs of instances in the two clusters $C_k$ and $C_{k'}$



**Centroid-Linkage**
The minimum distance

$$d(C_k, C_{k'}) = d(\boldsymbol{c}_k, \boldsymbol{c}_{k'})$$

of the two centroids $\boldsymbol{c}_k$ and $\boldsymbol{c}_{k'}$ for the two clusters $C_k$ and $C_{k'}$



## 4.3. Cluster Evaluation

**Within Cluster Score** is a measure of cluster compactness, i.e. how close elements of a cluster are. Given the no. of clusters $K$, the $k$-th cluster $C_k$, the $i$-th data point $\boldsymbol{x}_i$, the centroid $\boldsymbol{c}_k$, and the distance function $d(\cdot)$, we can find it by:

$$\mathrm{WC} = \sum_{k=1}^{K} \sum_{i=C_k} d(\boldsymbol{x}_i, \boldsymbol{c}_k)^2$$

**Between Cluster Score** is a measure of cluster separation, i.e. how far two clusters are.
Given $d(\boldsymbol{c}_k, \boldsymbol{c}_{k'})$ as the distance between centroids of $C_k$ and $C_{k'}$, it is given by:

$$\mathrm{BC} = \sum_{k=1}^{K-1} \sum_{k'=k+1}^{K} d(\boldsymbol{c}_k, \boldsymbol{c}_{k'})^2$$

The **Calinski-Harabaz Index** is based on the sum of squared distances of the points to their closest centroids, it is high for when the clusters are well separated and dense.

It is relatively easy to compute, given the number of instances $n$, number of clusters $K$, between cluster sub of squared distances BC, and within cluster sum of squared distances WC:

$$\left(\frac{\mathrm{BC}}{\mathrm{WC}}\right)\left(\frac{n-K}{K-1}\right)$$

**Silhouette Coefficient**

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}} = \begin{cases} 1, & \text{for dense and well-separated clusters} \\ 0, & \text{for overlapping clusters} \\ -1, & \text{for incorrect clustering} \end{cases}$$

Where $a_i$ is the mean distance metric between $\boldsymbol{x}_i$ and all other data points in the same cluster $C_k$:

$$a_i = \frac{1}{m_k - 1} \sum_{\substack{i' \in C_k \\ \{i\}}} d(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$$

Where $b_i$ is the mean distance between an instance and all other points in the nearest cluster $C_{k'}$:

$$b_i = \frac{1}{m_{k'}} \sum_{i' \in C_{k'}} d(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$$

**Minimum Description Length (MDL)** evaluates how good a set of clusters is, by determining the amount of information, i.e. number of bits, required to store the clustered data

**Occam's Razor**: the best model is the simplest

With clustering, we need to represent:
- Number of clusters.
- Membership description for each cluster, e.g. centre.

Given a model/hypothesis $h$, number of bits to encode the model $\text{len}(h)$, a data set $D$, encoding of the data set using our model $\text{len}(D|h)$, we **minimise total description length** $\text{len}(h) + \text{len}(D|h)$.

Given training set $D$, we want to find the most likely model $h$ for the data. Thus, we want to maximize the posterior probability $\Pr(h|D)$ (probability model $h$ is correct given set of examples $D$). By using Bayes rule and ignoring $\Pr(D)$, we can minimise

$$\begin{aligned} -\log(\Pr(h|D)) &= -\log(\Pr(D|h) \cdot \Pr(h)) \\ &= -\log(\Pr(D|h)) - \log(\Pr(h)) \end{aligned}$$

## 4.4. Outlier Detection

**Outlier/Anomaly Detection** is identifying objects in the data different from most other objects.

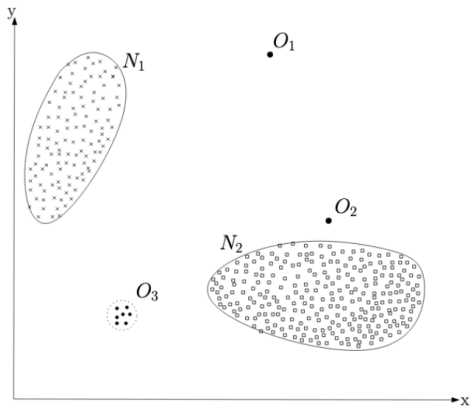



Figure 38: Clusters have outliers $O_1, O_2$ and outlier region $O_3$

Outlier detection techniques can either be supervised (data set manually labelled), semi-supervised (collecting data is not easy, partially labelled data), or unsupervised (no labelled data is available).

**Model-based Outlier Detection** is where we find observations that differ so much from other observations to arouse suspicion that it has been generated by a different mechanism; outliers do not fit well the model of the data (e.g. regression model, classifier, cluster set)

**Proximity-based Outlier Detection** is where we look for objects that have a very high distance from its $k$-nearest neighbour:

$$O = \left\{ \boldsymbol{x}_i : d_{k(\boldsymbol{x})} \geq T, 1 \leq i \leq n \right\}$$

This requires defining a proximity measure between instances and computing objects which are distant from most other objects. We define distance function $d_{k(\boldsymbol{x})}$ and threshold value $T$.

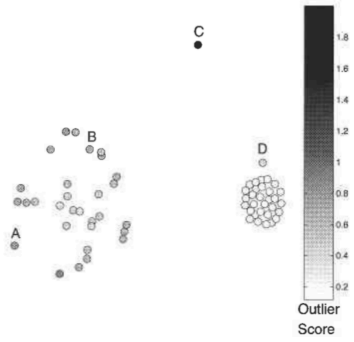
**Density-based Outlier Detection**

This is the inverse of the average distance to the $k$-nearest neighbors.

$$\delta_k(\boldsymbol{x}) = \left( \frac{1}{|N_{k(\boldsymbol{x})}|} \sum_{\boldsymbol{x}' \in N_{k(\boldsymbol{x})}} d(\boldsymbol{x}, \boldsymbol{x}') \right)^{-1}$$

Where $d(\boldsymbol{x}, \boldsymbol{x}')$ is the distance between $\boldsymbol{x}$ and $\boldsymbol{x}'$ and $N_{k(\boldsymbol{x})}$ is the set of $k$-nearest neighbours of $\boldsymbol{x}$. The density is an outlier score, if it is very small, then $\boldsymbol{x}$ is an outlier.

If the data set contains regions with different densities, then finding outliers might not be possible.




We can deal with this by computing the relative density, based on average density of neighbours:

$$r_{k(\boldsymbol{x})} = \frac{\delta_{k(\boldsymbol{x})}}{\frac{1}{|N_{k(\boldsymbol{x})}|} \sum_{\boldsymbol{x}' \in N_{k(\boldsymbol{x})}} \delta_{k(\boldsymbol{x}')}}$$

**Cluster-based Outlier Detection**
Simply consider any object that does not strongly belong to a cluster as an outlier.
If the elimination of an object results in a substantial improvement in the cluster evaluation objective, then the object is considered to be an outlier.

Outliers may heavily depend on the number of clusters.
If a data point is isolated even with a large number $K$ of clusters, then it is more likely to be a true outlier.

**Probabilistic Outlier Detection**
A probabilistic model specifies way of computing a probability that each a data point appears in the data set. If the probability of a data point xi is below a threshold value $T$, then $x_i$ is an outlier.

# 5. Statistical Modelling

Probabilities/statistics are useful for:

- Building linear regression (e.g. assumption of normally distributed residuals)/classification (e.g. computing class with highest prob., determining good splits in decision trees) models.
- Training models with probabilistic framework: maximum likelihood estimation, Bayesian est.
- Evaluating models

## 5.1. Probability Distributions

Probability distributions model uncertainty, e.g. future events (stock value) or hidden variable values (disease from symptoms). Data is a limited sample of problem under consideration. It is used to quantify our degree of belief that a certain event will occur.

A **random variable** $x$ represents an object that can take one among the set of possible values, $\text{dom}(x)$, the domain of $x$

If $x$ is discrete, then $\text{dom}(x) = \{v_1, v_2, ..., v_m\}$ is a discrete set of discrete values.

A **probability distribution** is a probability that each value in $\text{dom}(x)$ occurs for some random variable $x$; $\Pr(x = v_i)$ = probability that a single value $v_i \in \text{dom}(x)$ occurs

A **cumulative distribution function** is the probability $\Pr(x \leq v_i)$ for each $v_i \in \text{dom}(x)$ for some random variable $x$

**Example: birthdays of a population**

Suppose $x$ is a random variable representing month of birth.

The domain of $x$, $\text{dom}(x) = \{\text{Jan}, \text{Feb}, ..., \text{Dec}\}$.

$\Pr(\text{Oct}) = \frac{1}{12}, ...$

The cumulative distribution is:

$$\begin{aligned}\Pr(x \leq \text{Mar}) &= \Pr(\text{Jan}) + \Pr(\text{Feb}) + \Pr(\text{Mar}) \\ &= \frac{1}{12} + \frac{1}{12} + \frac{1}{12} \\ &= \frac{1}{4}\end{aligned}$$

### 5.1.1. Joint Distributions

A **multivariate random variable** is a set of random variables $x_1, ..., x_n$ where each variable $x_i$ has its own distribution

A **joint distribution** is combination of all component distributions, e.g. given a multivariate random variable

If there is no relationship between a set of random variables, they are called **independent**.

If two (or more) random variables are related to each other, then they are called **dependent**.

- Correlation is not the same as statistical dependnece.
- Two independent variables must be non-correlated.
- Two dependent variables are correlated or non-correlated.
- Correlation does not imply causation.
- Causation is captured by statistical (in)dependence.

### 5.1.2. Populations and Samples

Often, a data set is a sample from a population.
Given a representative sample (one that accurately reflects characteristics of the population), *statistical inference* allows making statements about the population using the sample. Each member of pop. has a probability of appearing in the sample.

A major data mining goal is determining the best **hypothesis**, i.e. candidate model, in a space of possible hypotheses for making predictions.

Hypotheses we could make about a population, built from a sample:
- $\Pr(D)$: probability of observing data $D$ without hypotheses
- **Prior** $\Pr(h)$: initial probability that hypothesis $h$ holds without having observed any data
- **Likelihood** $\Pr(D|h)$: probability of observing data $D$, given world where hypothesis $h$ is true
- **Posterior** $\Pr(h|D)$: probability hypothesis $h$ is true, givne we have observed data set $D$

### 5.1.3. Likelihoods

A **random trial** is a procedure with a set of well-defined outcomes. Often, we aim in making predictions about the overall outcome of a series of trials.

The **binomial distribution, $B(n, p)$,** is a distribution of random variable, which counts number of successes in a sequence of $n$ independent trials, where each trial asks a yes/no question; a trial produces the outcome 'yes' with probability $p$ and therefore 'no' with probability $1 - p$

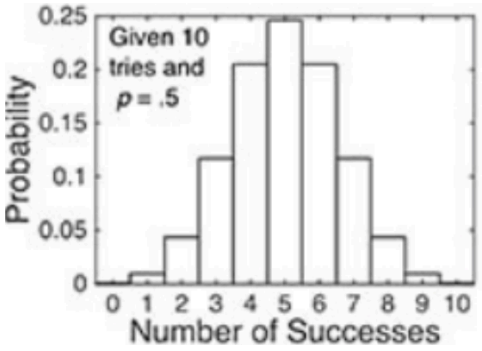


Figure 39: Bionomial Distribution

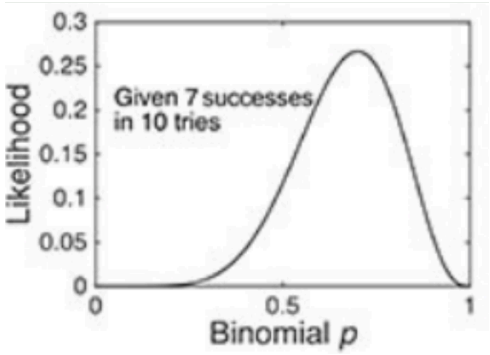


Figure 40: Binomial Likelihood Function

#### 5.1.4. Estimation

Estimation can be used to:

- allow incoporating knowledge and/or beliefs that we have in the model before observing the data
- enable revising and improving the model when new data becomes available
- can be computed in different ways

The **Maximum Likelihood Estimation (MLE)** computes the parameters of $h$ so that the likelihood $L(h|D)$ is maximised, i.e. $\arg\max_h\{L(h|D)\}$.

Assume parameters have unknown but fixed values. Defining likelihood for a given problem requires specifying a probabilistic model of how the data was generated.

Consider a data set $D = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_m\}$ assuming independence of observations:

$$\begin{aligned} L(h|D) &= \Pr(D|n) \\ &= \Pr(\boldsymbol{x}_1, ..., \boldsymbol{x}_m \mid h) \\ &= \Pr(\boldsymbol{x}_1 \mid h) \cdot \Pr(\boldsymbol{x}_2 \mid h) \cdot ... \cdot \Pr(\boldsymbol{x}_m \mid h) \\ &= \prod_{i=1}^{m} \Pr(\boldsymbol{x}_i \mid h) \end{aligned}$$

The **Bayesian Estimation** computes a model $h$ of maximum posterior probability $\Pr(h|D)$, i.e. $\arg\max_h\{\Pr(h \mid D)\}$.

Given a set of mutually exclusive candidate hypotheses whose probabilities sum to 1, we find the most probable for the data. Computes prior probability distributions of $h$ parameter values, rather than a single value. Computes posterior probability with Bayes rule:

$$\Pr(h \mid D) = \frac{\Pr(D|h) \cdot \Pr(h)}{\Pr(D)}$$

## 5.2. Naïve Bayes Classification

**Naïve Bayes Classification** is a probabilistic classification technique that uses Bayes Rule; assume conditional independence of attributes (unrelatistic/naïve for real-world data sets but often works well in practice)

A data distribution can be computed with proportions of instances in the data set, this information can be used to identify the most likely among different hypotheses.

**Bayes Rule of Conditional Probability**
Given a hypothesis $h$ that something will occur and new data $d$ which is observed, the probability that $h$ will be true given $d$ is:

$$\Pr(h|d) = \frac{\Pr(d|h)\Pr(h)}{\Pr(d)}$$

The **conditional independence assumption** is that attributes are conditionally independent given the class; combined probability can be obtained by multiplying individual probabilities

> **Note**
>
> This assumption may not hold if attributes are correlated, but we can work around this by carefully selecting the attributes that we use.

INCOMPLETE come back to this in the future

### 5.2.1. Gaussian/normal distributions

Numeric attributes can be modeled with a Gaussian (normal) $N(\mu, \sigma^2)$ distribution. Instead of computing likelihoods by counting occurrences of attribute values, we may use the mean $\mu$, standard deviation $\sigma$ and probability density function (pdf) $f(\cdot)$ of the assumed Gaussian distribution.

The **probability density function**, $f(\cdot)$, is the probability that an attribute $x$ falls within a region $\Pr(x - \varepsilon \leq x \leq x + \varepsilon) = \varepsilon \cdot f(x)$

**Probability Distribution Function for Normal Distribution**

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{(x-\mu)^2}{\sigma^2}\right)}$$

**Example: For Table 3, find probability temperature is 66 given play is 'yes'**

Weather data set with two numeric attributes (temperature, humidity) and two nominal attributes (outlook, windy).

For all instances with play = 'yes', $\mu = 73$ and $\sigma = 6.2$:

$$\begin{aligned} f(\text{temp} = 66|\text{yes}) &= \frac{1}{6.2\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{(66-73)^2}{6.2^2}\right)} \\ &= 0.0340 \end{aligned}$$

Using this value, after dropping $\Pr(d)$, we may compute the posterior probability:

$$\begin{aligned} \Pr(\text{yes}|d) &= \frac{2}{9} \cdot 0.0340 \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{9}{14} \\ &= 0.0008 \end{aligned}$$

### 5.2.2. Laplace estimation

It may be problematic if some possible attribute values are missing from the training set, e.g. no examples for 'misty' given 'yes': $\Pr(\text{misty}|\text{yes}) = 0$. We can deal with this using Laplace estimation.

**Laplace Estimation**

In the computation of likelihoods, for each possible attribute value, add 1 to the numerator and $\ell$ to the denominator, where $\ell$ is the number of attribute values.

**Example of Laplace Estimation**

For example, in the case of outlook, there are 3 possible values in the data set: sunny, overcast, and rainy.

The probabilities would be modified as follows:

- $\Pr(\text{sunny}|\text{yes}) = \frac{2}{9} \to \frac{2+1}{9+4} = \frac{3}{13}$
- $\Pr(\text{overcast}|\text{yes}) = \frac{4}{9} \to \frac{4+1}{9+4} = \frac{5}{13}$
- $\Pr(\text{rainy}|\text{yes}) = \frac{3}{9} \to \frac{3+1}{9+4} = \frac{4}{13}$
- $\Pr(\text{misty}|\text{yes}) = \frac{0}{9} \to \frac{0+1}{9+4} = \frac{1}{13}$

## 5.3. Density Estimation

If $x$ is a continuous random variable, then its domain $\text{dom}(x) = [v_{\min}, v_{\max}]$ is a continuous range.

A continuous random variable is associated with a **probability density function** $f$ which allows computing the probability that $a \leq x \leq b$, for some values $a, b \in \text{dom}(x)$.

$$\Pr(a \leq x \leq b) = \int_a^b f(u).du$$

**Cumulative distribution function** $F(a) = \Pr(x \leq a)$

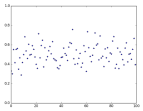
There exists various distributions other than the normal:

- Uniform
- Poisson
- Exponential
- Gamma

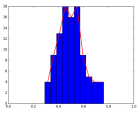
Because of the **central limit theorem** when independent random variables are added, their properly normalized sum often tends to a normal distribution.

**Density estimation** is, given a data set, computing an estimate of an underlying probability density function. It can either be:

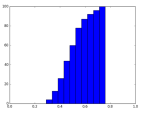
- parametric: select common distribution (e.g. gaussian) and estimate parameters (mean, std)
- non-parametric: fit a model to arbitrary data distribution (e.g. kernel density estimation)



(a) 100 random numbers chosen from a **Gaussian** distribution



(b) probability density



(c) cumulative probability

**Multivariate Guassian (Normal) Distribution**

The probability density function for $n$-dimensional variables is:

$$f(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{n}{2}} \ |\sum|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \sum^{-1} (\boldsymbol{x}-\boldsymbol{\mu})}$$

- $\mu$: $n$-dimensional vector of the means
- $\sum$: the $n \times n$ covariance matrix with determinant $|\sum|$
- $(2\pi)^{\frac{n}{2}} \ |\sum \frac{|^1}{2}$: normalise constant to ensure that $f(\boldsymbol{x})$ integrates to 1 (valid pdf)
- $(\boldsymbol{x} - \boldsymbol{\mu})^T \sum^{-1} (\boldsymbol{x} - \boldsymbol{\mu})$: **Mahalanobis distance** between $\boldsymbol{x}$ and $\boldsymbol{\mu}$, often denoted by $r^2_{\sum}(\boldsymbol{x}, \boldsymbol{\mu})$
  Generalisation of Euclidean distance by accounting, through $\sum$, for correlations

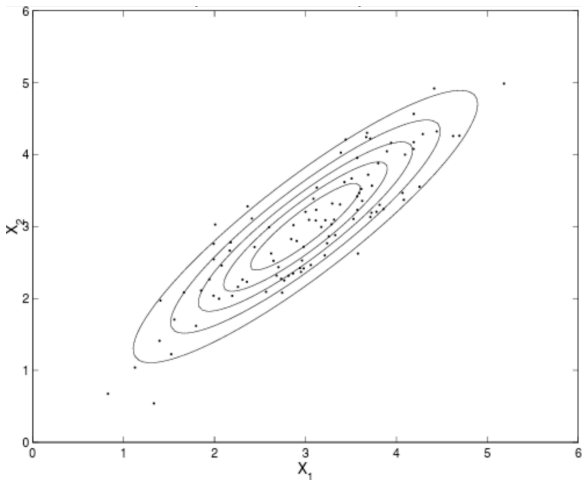
> **Note**
>
> We can simplify this for a normal model.
> - Use the shorter notation for Mahalanobis distance.
> - Replace the normalisation by simple constant $K$.
>
> $$f(\boldsymbol{x}) = \frac{1}{K} e^{-\frac{1}{2} r^2_{\sum}(\boldsymbol{x}, \boldsymbol{\mu})}$$

**Iso-Density Contours** are all points $x$ with equal density $f(\boldsymbol{x}) = c$, i.e. the same $r^2_{\sum}(\boldsymbol{x}, \boldsymbol{\mu})$ value for some constant $c$

For a multi-variate normal distribution, they trace out an ellipse with center at $\mu$, and height falling exponentially from the center as a function of $r^2_{\sum}(\boldsymbol{x}, \boldsymbol{\mu})$.

Example: iso-density contours illustration

Suppose a multi-variate normal model where $n = 2$, $\boldsymbol{\mu} = [3, 3]$, and $\sum = \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix}$.




The **eccentricity** and **orientation** of a multivariate normal model elliptical contours are determined from $\sum$:

- If $\sum$ is the **identity matrix** (diagonals are 1, 0 rest), then all variables have the same variance, are uncorrelated, and the contours are circles.
- If $\sum$ is the **diagonal matrix** (all non-diagonal elements are 0), then the contours are parallel to the variables axes and elongated along the axes with the greater variance.
- If $\sum$ indicates **high correlation between some variables**, then the elliptical contours are elongated along vectors defined as linear combinations of these variables.

**Independence Assumption**
For high-dimensional data with a large number $n$ of variables and fixed number $m$ of data points, there are many terms in $\sum$.

Often, we may only obtain unreliable covariance estimates and we may not want to model all these terms explicitly. Assuming variable independence (i.e. diagonal $\sum$), multivariate normal density can be expressed as a product of univariate normal densities. (this can be a *strong assumption*)

**Block Diagonal Covariance Matrix Assumption**
Variables can be divided into groups/blocks. Variables in the same block are dependent, but variables across groups are independent. Two variables are *conditionally independent* given the other variables, if any only if the corresponding $\sum^{-1}$ element is zero. Hence $\sum^{-1}$ reveals pattern relationships between variables.

The **mean** of an attribute $j$ is:

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_{i,j}$$

Where $x_{i,j}$ is an attribute $j$ value for instance $i$, $m$ is the number of instances, and $n$ is the number of variables/attributes. An attribute is considered to be a random variable.

The mean provides information about a population given a sample of observations, i.e. a data set.

> **Note**
>
> It is a 'central statistical measure' in the sense that it minimises:
>
> $$f(y) = \sum_{i=1}^{m} \left(y - x_{i,j}\right)^2$$

The **variance** of an attribute $j$ is the average deviation between data values and the mean:

$$\mathrm{Var}_j = \frac{1}{m}\sum_{i=1}^{m} \left(x_{i,j} - \mu_j\right)^2$$

Indicates how attribute values are spread out from their mean. Good indicator of outliers.

The **standard deviation** of an attribute $j$ measures the spread around mean of the data:

$$\begin{aligned}\sigma_j &= \sqrt{\mathrm{Var}_j} \\ &= \sqrt{\frac{1}{m}\sum_{i=1}^{m} \left(x_{i,j} - \mu_j\right)^2}\end{aligned}$$

- Measured in same units as original $x_{i,j}$ scale.
- Useful for comparing two data sets with the same mean.

The **covariance** of two attributes $j$ and $j'$ is the statistical measure of relationship between them:

$$\mathrm{Cov}_{j,j'} = \frac{1}{m}\sum_{i=1}^{m}(x_{i,j} - \mu_j)\left(x_{i,j'-\mu_{j'}}\right)$$

Measures the joint variability of two random variables. Value is positive if large values of $j$ are associated with large values of $j'$ and negative if large values of $j$ are associated with small values of $j'$.

We can build the **covariance matrix** for an arbitrary no. of attributes:

$$\boldsymbol{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$$

e.g. for three attributes, this produces

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} & \sigma_{1,3} \\ \sigma_{2,1} & \sigma_{2,2} & \sigma_{2,3} \\ \sigma_{3,1} & \sigma_{3,2} & \sigma_{3,3} \end{bmatrix}$$

## 5.4. Mixture Models & Gaussian Mixture Models

In practice, large data sets are typically **heterogeneous**: may capture different underlying phenomena of data collected to form one large data set and may represent multiple subpopulations or groups, than a single homogeneous population.

A **mixture distribution** is a probability distribution derived as a collection of simpler distributions.

Mixture distributions allow us to represent subpopulations of a larger population without requiring the observed data to specify the subpopulation to which an instance belongs.

The **Poisson Distribution** models the number of events occuring per time, using the p.d.f.:

$$f(\boldsymbol{x}) = \frac{(\lambda)^x e^{-\lambda}}{x!}$$

Where $\lambda$ is the rate at which events occur and $x$ is a random variable corresponding to no. of events.

**Example mixture model problem**

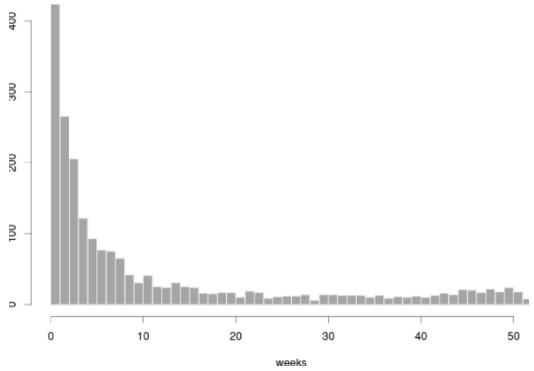



Figure 42: Histogram of supermarket purchases with a specific credit card type during the year 1996

The credit card data set appears to be bimodal with:

- a large mode to the left for 7-10 weeks
- a smaller mode to the right for 42-45 weeks

One Poisson model would fail to capture this, instead we can model the empirical distribution with two components, i.e. assume there are two kinds of people: those who are unlikely to use their card in a supermarket and those who use it most weeks.

We define two parameters $\lambda_1, \lambda_2$, and a probaility one belongs to the first group, $\pi$:

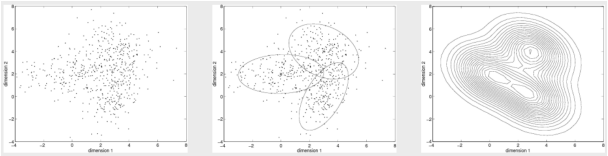
$$f(\boldsymbol{x}) = \pi \cdot \frac{(\lambda_1)^x e^{-\lambda_1}}{x!} + (1-\pi) \cdot \frac{(\lambda_2)^x e^{-\lambda_2}}{x!}$$

This is what we call a mixture model.

A **mixture model** consists of multiple component models each one specified by its own parameters:

$$f(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k f_k(\boldsymbol{x}; \boldsymbol{w}_k)$$

Where $K$ is the number of components/densities
Where $f_k(\boldsymbol{x}; \boldsymbol{w}_k)$ is a distribution of component $k$, e.g. simple Gaussian w. well-defined parameters
Where $\boldsymbol{w}_k$ are parameters of component $k$
Where $\pi_k$ is weight of component $k$, i.e. probability a data point is generated by component $k$
This value must be $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$



(a) data points generated from three distributions (b) underlying component densities within $\mu \pm 3\sigma$ (c) contours of overall mixture density function

Figure 43: Mixture Distribution Illustration

> **Note**
>
> Mixture models can be used for clustering:
> - Associate a component parametric model to each cluster
> - Assign each data point to the cluster that it most likely comes from

The **log-likelihood of a mixture model** is given by:

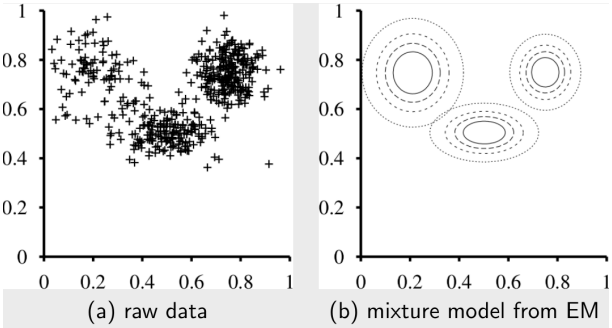
$$L(\boldsymbol{\pi}, \boldsymbol{w}_1, ..., \boldsymbol{w}_K) = \log \left[ \sum_{k=1}^{K} \pi_k f_k(\boldsymbol{x}; \boldsymbol{w}_k) \right]$$

To compute a mixture model for a data set, we may maximize log-likelihood leading to a non-linear optimization problem.

A **Gaussian Mixture Model (GMM)** is a mixture model with Gaussian components $C_1, ..., C_K$:

$$\Pr(\boldsymbol{x}_i) = \sum_{k=1}^{K} \Pr(C_k) \Pr(\boldsymbol{x}_i \mid C_k)$$

Where $\boldsymbol{x}_i$ is the data point/instance, $C_k$ is the $k$-th Gaussian component, $\Pr(C_k)$ is the probability of $C_k$, $\mu_k$ is the mean, and $\sigma_k$ is the standard deviation.




(a) raw data (b) mixture model from EM

Figure 44: Mixture Model Illustration

### 5.4.1. EM Algorithm

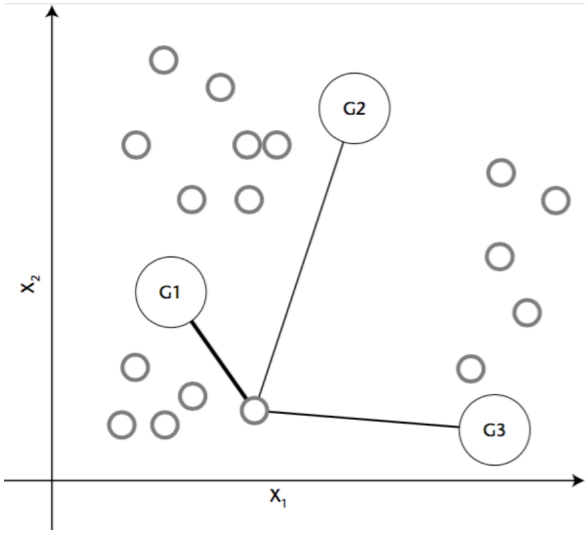
The **Expectation-Maximization (EM) Algorithm** is a well-known algorithm for computing GMMs which aims to maximise log-likelihood. We repeat the two steps:

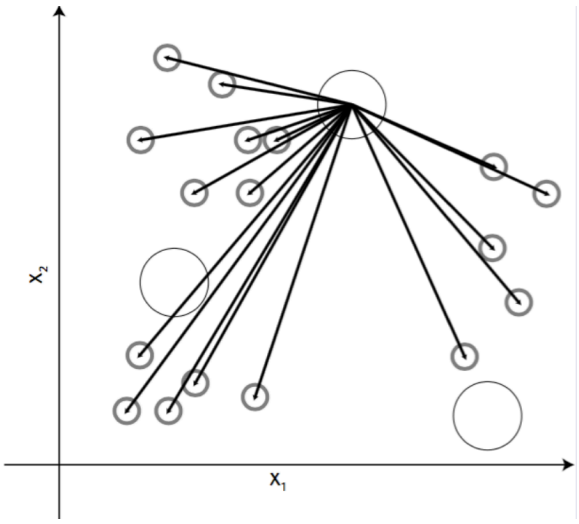
**Expectation**

Decide a Gaussian component $C_k$ that each data point $\boldsymbol{x}_i$ belongs to; a Gaussian is more responsible for points close to its mean and less responsible for more distant points

**Maximisation**

Compute new parameters $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and weights $\pi_k$ for each component $C_k$; i.e. compute means and covariances using weights from previous step




**Expectation-Maximization (EM) Algorithm**

1. Compute probability $\pi_{i,k} = \Pr(C_k|\boldsymbol{x}_i)$ that data point $\boldsymbol{x}_i$ has been generated by $C_k$
2. Bayes rule, compute $\Pr(\boldsymbol{x}_i|C_k)\Pr(C_k)$
3. $m_k = \sum_{i=1}^{m} \pi_{i,k}$
   no. of points assigned to $C_k$

1. Compute new mean
$$\boldsymbol{\mu}_k \leftarrow \sum_{i=1}^{m} \left(\frac{\pi_{i,k}}{m_k}\right) \boldsymbol{x}_i$$
2. Compute new covariances
$$\boldsymbol{\Sigma}_k \leftarrow \sum_{i=1}^{m} \left(\frac{\pi_{i,k}}{m_k}\right) (\boldsymbol{x}_i - \boldsymbol{\mu}_j)(\boldsymbol{x}_i - \boldsymbol{\mu}_j)^T$$
3. Compute new component weight
$$\pi_k \leftarrow \frac{m_k}{m}, \quad \text{i.e. } \Pr(C_k)$$

**EM Algorithm Properties**

- *Hill-climbing algorithm* in the multivariate parameter space, where the direction and distance are specified in the two steps.
- Does not decrease log-likelihood in any step.
- Takes initially big steps and then converges more slowly.
  Often, convergences in 5-20 iterations.
- Running EM with several different starting points is important.
- If the standard deviation of a component is 0, then the likelihood may become to $+\infty$. E.g. this may happen for a component with only one data point. It may be useful to constrain the model, e.g. enforce all standard deviations to be equal and non-zero.

# 6. Instance-Based Learning

**Instance-based classification** works by, instead of creating models, using the actual instances for knowledge representation. Hence, the main work is done when classifying new instances as we search the memory for the training instance most resembles the new one.

There are two main components:
- distance function: calculates distance between any two instances
- combination function: combine results from neighbours to arrive at an answer

An instance-based model requires a **distance metric** which compares new instances to existing ones. We may use weighting schemes when some attributes are more important than others.
- Numeric attributes: use arithmetic difference

$$
\begin{aligned}
|x_{i,j} - x_{i',j}| &\quad \text{absolute difference} \\
\left(x_{i,j} - x_{i',j}\right)^2 &\quad \text{squared difference} \\
\frac{|x_{i,j} - x_{i',j}|}{\max_j - \min_j} &\quad \text{normalised absolute difference} \\
\frac{|x_{i,j}| - \mu_j}{\sigma_j} - \frac{x_{i',j} - \mu_j}{\sigma_j} &\quad \text{absolute difference of standardised values}
\end{aligned}
$$

- Nominal attributes: check whether values are identical/different, other. sophisticated comparison

$$
d(x_{i,j}, x_{i',j}) = \begin{cases} 0, \text{if } x_{i,j} = v \text{ and } x_{i',j} = v \\ 1, \text{if } x_{i,j} = v \text{ and } x_{i',j} \neq v \end{cases}
$$

- Distance between instances (aggregation across attributes):

$$
\begin{aligned}
d(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) &= \sqrt{\sum_{j=1}^{n} \left(x_{i,j} - x_{i',j}\right)^2} \quad \text{Euclidean distance} \\
d(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) &= \sum_{j=1}^{n} |x_{i,j} - x_{i',j}| \quad \text{Manhattan distance}
\end{aligned}
$$

> **Note**
>
> Example: memory-based reasoning, use past experience
> - Medical treatment: what treatment plans had positive outcomes for similar patients?
> - Fraud detection: is a new case similar to previous fraud?
> - Customer response prediction: is a new customer similar to customers who responded pos.?

**Advantages/Disadvantages of instance-based learning**
- +‘ve: Instance-based methods carve boundaries separating the data set into neighbourhoods.
- neut: Discarding non-representative class instances may be required for efficiency.
- +‘ve: not restricted by data format
- +‘ve: efficiently adaptable to addition of new instances
- +‘ve: requires relatively low training effort
- -‘ve: memory intensive because large amount of historical data must be readily available
- -‘ve: generally more time-consuming
- -‘ve: finding suitable distance/combination functions is challenging

## 6.1. Basic Elements & Applications

**Example of marketing data set with customers**

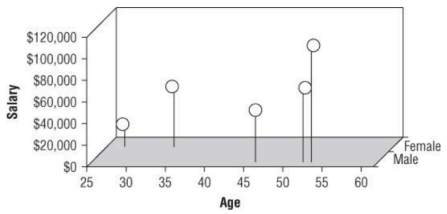
Suppose the data:

| RECNUM | GENDER | AGE | SALARY |
|---|---|---|---|
| 1 | Female | 27 | $ 19,000 |
| 2 | Male | 51 | $ 64,000 |
| 3 | Male | 52 | $105,000 |
| 4 | Female | 33 | $ 55,000 |
| 5 | Male | 45 | $ 45,000 |

We can compute distance by using one numeric attribute, e.g. age:

| | 27 | 51 | 52 | 33 | 45 |
|---|---|---|---|---|---|
| **27** | 0.00 | 0.96 | 1.00 | 0.24 | 0.72 |
| **51** | 0.96 | 0.00 | 0.04 | 0.72 | 0.24 |
| **52** | 1.00 | 0.04 | 0.00 | 0.76 | 0.28 |
| **33** | 0.24 | 0.72 | 0.76 | 0.00 | 0.48 |
| **45** | 0.72 | 0.24 | 0.28 | 0.48 | 0.00 |

We can combine each attribute distance into a single distance function, obtaining a distance between every pair of instances. A scatter plot in three-dim illustrates distances between instances:



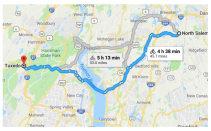

We can aggregate distance values across attributes to obtain nearest neighbours:

| | $D_{SUM}$ | $D_{NORM}$ | $D_{EUCLID}$ |
|---|---|---|---|
| **1** | 1,4,5,2,3 | 1,4,5,2,3 | 1,4,5,2,3 |
| **2** | 2,5,3,4,1 | 2,5,3,4,1 | 2,5,3,4,1 |
| **3** | 3,2,5,4,1 | 3,2,5,4,1 | 3,2,5,4,1 |
| **4** | 4,1,5,2,3 | 4,1,5,2,3 | 4,1,5,2,3 |
| **5** | 5,2,3,4,1 | 5,2,3,4,1 | 5,2,3,4,1 |

Here, the sets of nearest neighbors are the same regardless of the way the distances are combined. This is coincidence because there are two well-defined clusters in the data set: (i) younger females and (ii) older males.

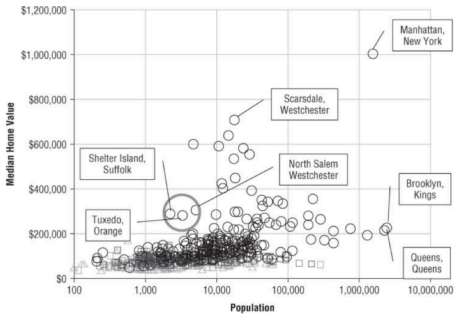
## Case Study: estimating rent costs


(a) Tuxedo - Shelter Island
150 miles




(b) Tuxedo - North Salem
50 miles

Estimate cost of renting an apartment in a target town by combining data of rents in similar towns (nearest neighbours – not necessarily geographic neighbours).




1. Identify nearest neighbours; Brooklyn and Queens geographically and on scatter plot; Tuexdo is close to Shelter Island and North Salem along the scatter plot dims. but these towns are miles apart.
2. Combine information from neighbours to make estimation about target town.

| TOWN | POP. | RENTING HOUSE-HOLDS | MEDIAN RENT | RENT <$500 | RENT $750 | RENT $1000 | RENT $1,500 | RENT >$1,500 | NON-CASH |
|---|---|---|---|---|---|---|---|---|---|
| Shelter Island | 2,228 | 160 | $804 | 3.1% | 34.6% | 31.4% | 10.7% | 3.1% | 17.0% |
| North Salem | 5,173 | 244 | $1,150 | 3.0% | 10.2% | 21.6% | 30.9% | 24.2% | 10.2% |
| *Tuxedo* | *3,334* | *349* | *$907* | *4.6%* | *27.2%* | *29.6%* | *23.8%* | *3.8%* | *14.8%* |

Could use: mean of neighbours median rents, mean of neighbour average rents, weighted average on no. of apartments, weighted average by distance, weighted average by distance and no. of apartments, etc. Many sensible approaches exist.

NB. calculations omitted here.

## Case Study: customer attrition

Attribute indicates whether the customer is inactive:

| RECNUM | GENDER | AGE | SALARY | INACTIVE |
|---|---|---|---|---|
| 1 | Female | 27 | $19,000 | no |
| 2 | Male | 51 | $64,000 | yes |
| 3 | Male | 52 | $105,000 | yes |
| 4 | Female | 33 | $55,000 | yes |
| 5 | Male | 45 | $45,000 | no |
| New | Female | 45 | $100,000 | ? |

Determine whether new customer is active or inactive based on nearest neighbours, e.g. use Euclidean distance function and different no. of $k$ neighbours: (question mark = tie)

| | NEIGHBORS | NEIGHBOR ATTRITION | K = 1 | K = 2 | K = 3 | K = 4 | K = 5 |
|---|---|---|---|---|---|---|---|
| $d_{sum}$ | 4,3,5,2,1 | Y,Y,N,Y,N | yes | yes | yes | yes | yes |
| $d_{Euclid}$ | 4,1,5,2,3 | Y,N,N,Y,Y | yes | ? | no | ? | yes |

**Case Study: Shazam music recognition**

Mobile application that listens to a song and, after a few seconds of listening, identifies the song and provides information about it, e.g. name, artist, etc. Algorithm can be described as nearest neighbour approach. Challenges include background noise, microphone quality, and matching any snippet of a song.

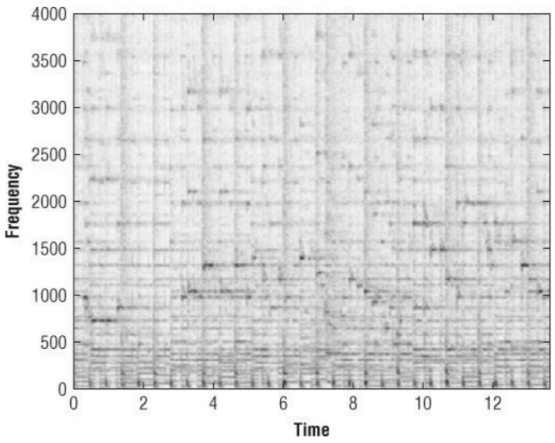



Figure 54: Spectrogram represents songs and snippets in the frequency domain

Remarks:

- vertical axis corresponds to music frequencies
- stronger frequencies are darker in chart
- horizontal axis represents time
- frequencies sampled every half second
- uniquely indentify a song — every two sepctrograms of same song unlikely to differ
- can be transformed into a constellation plot

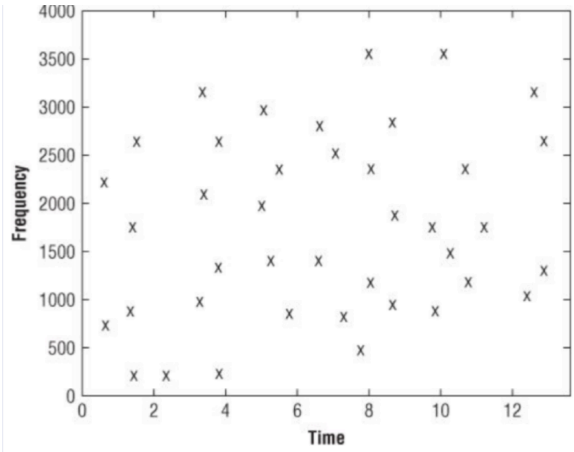



Figure 55: Constellation plot is a picture of the frequency peaks for a song (i.e. freq domain)

Remarks:

- each song is defined by unique constellation of peaks in frequency domain
- background noise does not have much effect on spectrogram peaks
- identifying a song becomes a problem of finding nearest neighbours for the constellation of a new song in the data set of known songs

Each instance is a list of peaks which has attributes: time into song, frequency, strength. Time and frequency are sufficient to characterise a song.

The metric for finding similar songs is therefore:

$$\frac{\text{number of peaks that the two songs have in common}}{\text{number of all distinct peaks}}$$

This works for complete songs but may not work for snippets.

- Instead we consider time slices and compare their peaks.
  - In a data set of millions of songs, this comparsion is intensive.
- Two audio files for the same song should overlap for sufficiently long and continuous time.

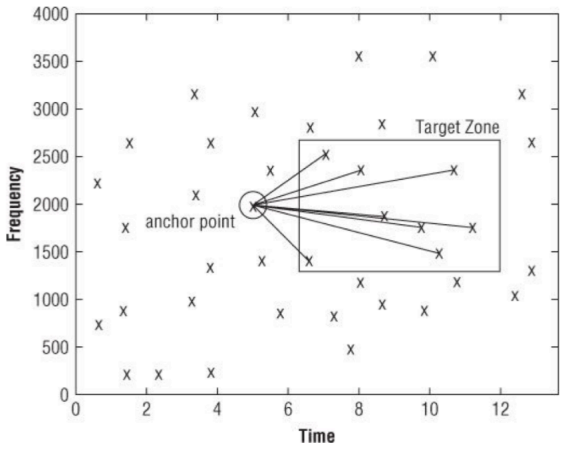



Figure 56: An anchor point is a peak in the constellation chart paired with other peaks that follow in the time axis within a specific range of frequencies.

The Shazam solution is to:

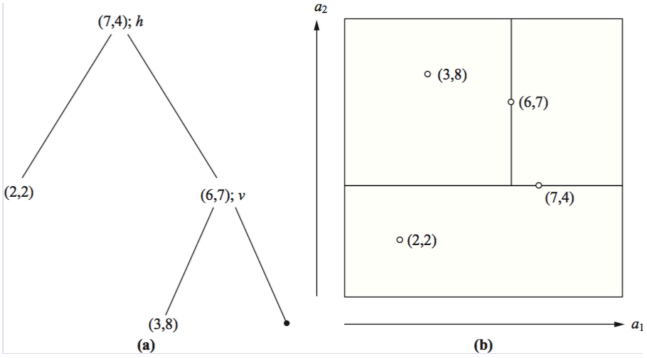
- associate each anchor-peak pair with:
  - time difference between peak and anchor
  - frequency difference between peak and anchor
- Use this information and nearest neighbour approach to check if something a known song and a new snippet are matched

Overall approach:

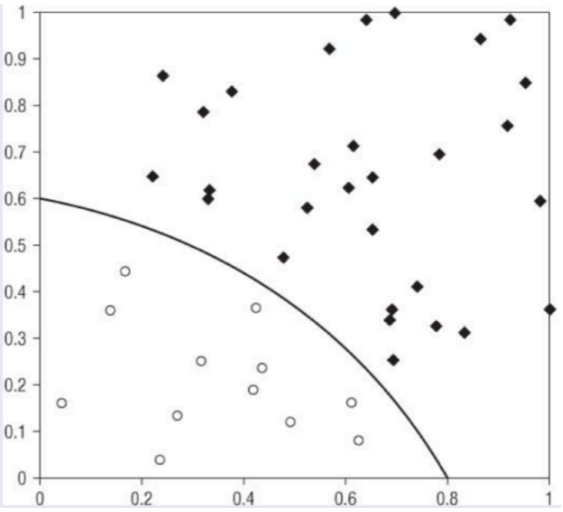
1. Convert the new snippet into a constellation plot.
2. Turn constellation into anchor points with anchor-peak pairs.
3. Identify matching anchor points between the listened song and songs in the database. Determine the longest consecutive sequence of overlap between the two.
4. Return the song with the longest overlap, i.e. the nearest neighbor

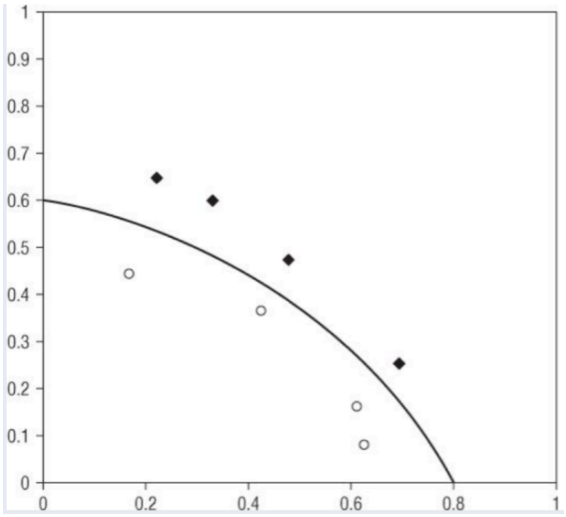
NN approaches are computationally intensive, especially when no. of training instances is large. We can try to mitigate this overhead by:

- using an efficient data structure, e.g. $k$D-trees




- reduce training set data size





  - classify each example w.r.t. known instances and save only misclassified instances
  - early discarded instances may turn out to be valuable
  - as number of stored instances increases, model accuracy increases
  - storing only misclassified is problematic in presence of noisy data

## 6.2. Non-Parametric Density Estimation

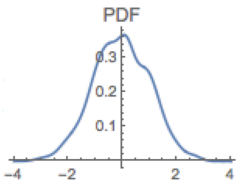
**Non-parametric models** are data-driven with few assumptions made about functional form and not characterised by a bounded set of parameters (histograms, nearest neighbours, kernel methods)

In the example of histograms, they can be problematic as:

- They provide a non-smooth, i.e. non-continuous and non-differentiable estimate of what is often presumed to be a smooth function.
- Choosing no. and position of bins is not obvious.
- Random fluctuation in small datasets can produce significantly different diagrams.

But despite their limitations they are useful for understanding data before modelling as they can reveal data quality problems, provide valuable information such as outliers and multimodality, and for large numbers of instances and relatively small number of attributes, histograms with small bin sizes may result in relatively smooth estimates.

A **kernel estimator** smooths contribution of a single data point around its neighbourhood. All points are retained in resulting model. (an attempt to improve histograms in a data-driven way)




The **kernel density estimate** is given by taking a local weighted average of measurements around the point $x$ of interest, given a kernel function $K(\cdot)$ and bandwidth of kernel $h$:

$$\hat{f}(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{K\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right)}_{\substack{\text{proportional to} \\ \text{sum of weights} \\ w_i}}$$

The kernel function is usually chosen to be smooth, unimodal, and should integrate to 1, i.e.

$$\int_{-\infty}^{+\infty} K(t).dt = 1$$

The **Gaussian kernel function** is a common pick:

$$K(t,h) = c \cdot e^{-\frac{1}{2}\left(\frac{t}{h}\right)^2}$$

Where $t$ is the distance between query point $x$ and $x_i$, $h$ is the bandwidth equivalent to $\sigma$ (std.), and $c$ is a normalisation constant.

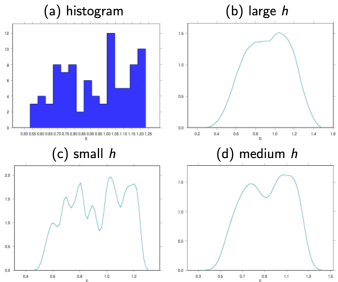



Figure 59: Choice of $h$ in a Gaussian kernel function determines smoothness of density function estimate. Large values lead to oversmoothing, small values lead to spiky estimates.

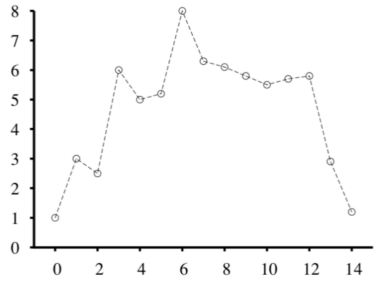
> **Note**
>
> - Kernel models are harder to define when number $n$ of attributes increases due to $n$-dim kern.
>   - One approach is to define $n$ one-dim kernels with bandwidths $h_1, ..., h_n$ but to keep the error constant as $n$ increases, we need the no. of data points to grow exponentially with $n$.
> - Kernel methods are memory-based so have significant memory requirements for large data.
> - Kernel models only work in practice for low-dimensional problems.
> - Kernel methods are closely related to nearest neighbour methods.

## 6.3. Locally Weighted Regression

A **regression** is, given a set $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)\}$ of numeric I/O pairs, learning $y = \hat{f}(\boldsymbol{x})$.
Parametric methods assume closed-form expression for $\hat{f}$ and compute params. of $\hat{f}$ min. squared E.
Non-parametric methods make very few assumptions and make assumptions in data-driven way.

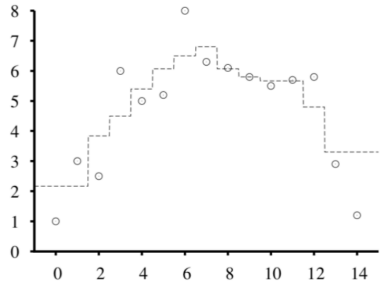
### Connect-The-Dots Regression

Compute a piecewise linear function connecting each data point $\boldsymbol{x}$ with two points on the left and right, respectively. Function is often spiky and does not generalise well. Trivial but common.



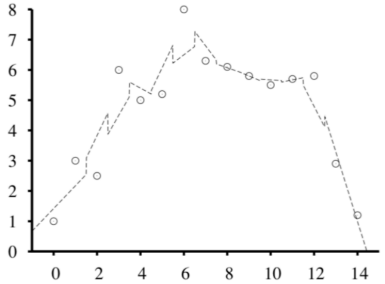

### $k$-Nearest Neighbours Regression

For each data point $\boldsymbol{x}$, compute mean $\hat{f}(\boldsymbol{x}) = \frac{1}{k} \sum_{i \in N_k(\boldsymbol{x})} y_i$ of data points in $N_k(\boldsymbol{x})$ (nearest neigh.)
Smooths magnitude of spikes but resulting function still has discontinuities.

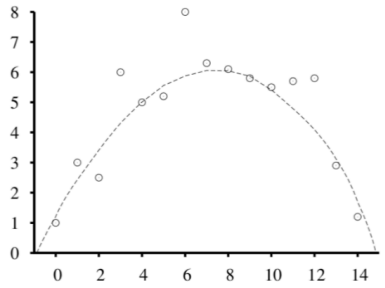



### $k$-Nearest Neighbours Linear Regression

For a new data point $\boldsymbol{x}$, compute linear regression line through set $N_k(\boldsymbol{x})$ ($k$-nearest neighbours).



**Locally Weighted Linear Regression** creates a function (model) with discontinuities. For each new data point $\boldsymbol{x}$, instances close to $\boldsymbol{x}$ are weighted heavily, while data points far away are weighted less heavily. We use a kernel to compute these weights.




Ordinary linear regression computes weights $\boldsymbol{w}$ minimising:

$$\sum_{i=1}^{m} (y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i)^2$$

Locally weighted linear regression does not compute a fixed set of parameters. Instead, parameters $\boldsymbol{w}$ are computed for each query point $\boldsymbol{x}$ by minimising:

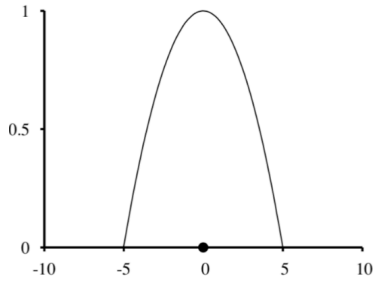
$$\sum_{i=1}^{m} K(d(\boldsymbol{x}, \boldsymbol{x}_i))(y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i)^2$$

Where $d(\boldsymbol{x}, \boldsymbol{x}_i)$ is the distance between query point $\boldsymbol{x}$ and training point $\boldsymbol{x}_i$, $K(\cdot)$ is the kernel function, and the answer to the query point is $\hat{f}(\boldsymbol{x}) = \boldsymbol{w}^* \cdot \boldsymbol{x}$ where $\boldsymbol{w}^*$ is the optimal vector of linear model weights at $\boldsymbol{x}$.

**Example Kernel Function**

Kernel function with bandwidth $h = 10$, centred on query point $\boldsymbol{x} = 0$:

$$K(d) = \max\left\{0, 1 - (2|\boldsymbol{x} - \boldsymbol{x}_i|/h)^2\right\}$$




A relatively small number of neighbours is often sufficient.

The weight of the kernel is higher in the centre and decreases as we move away.
Area below the curve should be bounded in $[-\infty, \infty]$.

Other kernel functions such as Gaussian can be used.
Though, deciding bandwidth is more important than actual function.

## 6.4. Support Vector Machines

**Support Vector Machines** are a non-parametric, kernel method for binary classification.
We use a kernel to construct a linear separator in higher-dimensional space (which is non-linear in original space). Construct a **maximum margin separator** — decision boundary with largest possible distance from data points.

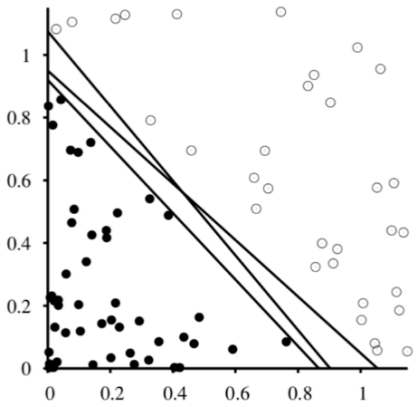



Figure 65: Three linear decision boundaries which are equally good in terms of error rate

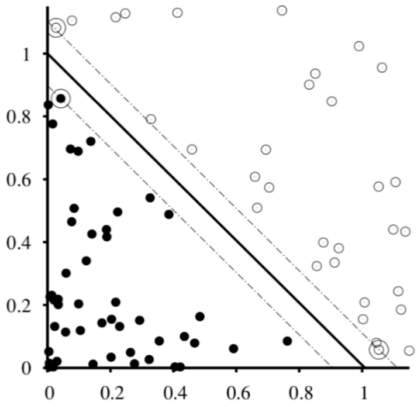



Figure 66: The maximum margin separator with support vectors on either side

- Typically, SVMs use class labels $+1$ and $-1$.
- A **linear separator** is a set of points $\{\boldsymbol{x} : \boldsymbol{w}\boldsymbol{x} + b = 0\}$.
- **Support vectors** are the points closest to the linear separator.
  They are called this as they are close, i.e. 'hold', the separating hyperplane.

**Computing Maximum Margin Separator**

- We can solve an optimisation problem by searching the space of $\boldsymbol{w}$ and $b$ with gradient descent to find all parameters that maximise margin while correctly classifying all examples.
- We can also solve the dual quadratic constrained optimisation problem (using duality):

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m-1} \sum_{i'=i+1}^{m} \alpha_i \alpha_{i'} y_i y_{i'} ; (\boldsymbol{x}_i \cdot \boldsymbol{x}_{i'})$$

$$\sum_{i=1}^{m} \alpha_i \boldsymbol{x}_i = 0$$

$$\alpha_i \geq 0$$

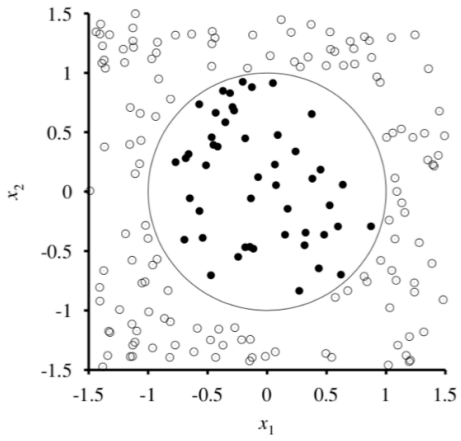
The optimisation problem parameters are number of data points, $m$, class of data point $\boldsymbol{x}_i$, $y_i$, and the dual variable, $\alpha_i$, associated with data point $\boldsymbol{x}_i$.
The objective function contains dot products of form $\boldsymbol{x}_i \cdot \boldsymbol{x}_{i'}$ and is convex.
Once vector $\alpha$ is computed, we can derive linear separator by setting $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \boldsymbol{x}_i$.
Hence, for a new data point $\boldsymbol{x}$, the actual linear separator is:

$$\hat{y} = \text{sign}\left( \sum_{i=1}^{m} \alpha_i y_i (\boldsymbol{x} \cdot \boldsymbol{x}_i) - b \right)$$

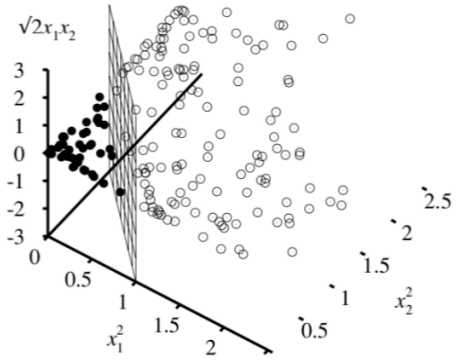
It is always possible to make data linearly separable by mapping them into a space of sufficiently high dimension, e.g. $m$ data points are always separable in spaces of $m - 1$ dimensions or more.

**Example mapping to higher-dimensional space**

Suppose the following data points, which clearly there is no linear separator for:




The true decision boundary is $x_1^2 + x_2^2 \leq 1$.
We can re-express the data by mapping each data point from $\boldsymbol{x} \in \mathbb{R}^2$ to $F(\boldsymbol{x}) \in \mathbb{R}^3$.




We can now use the following three features to linear separate the data:

$$f_1 = x_1^2$$
$$f_2 = x_2^2$$
$$f_3 = \sqrt{2}x_1x_2$$

Usually, we do not expect a linear separator in input space $\boldsymbol{x}$, but we can find linear separators in higher-dimensional feature space $F(\boldsymbol{x})$. We can therefore find linear separators by replacing $\boldsymbol{x}_i \cdot \boldsymbol{x}_{i'}$ with $F(\boldsymbol{x}_i) \cdot F(\boldsymbol{x}_{i'})$ in the quadratic optimisation problem.

The expression $K(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = F(\boldsymbol{x}_i) \cdot F(\boldsymbol{x}_{i'}) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_{i'})^2$ is a kernel function.

> **Note**
>
> SVMs extend the notion of perceptrons for computing linear separators but instead of training to minimise squared error, we try to maximise the margin between classes so as to achieve better generalisation.

# 7. Text Mining

## 7.1. Language Models

Basic elements of language models incl.:

- **characters**: individual symbols and building blocks of texts, e.g. letters, digits, punct. marks, spaces
- **string**: sequence of characters
- **language**: set of strings (words)
- **corpus**: collection of one or more texts

An ***n*-gram** is a contiguous sequence of $n$ items, e.g. characters.

Example: string $s = \text{abcdefg}$ is a 7-gram because it is $n = 7$ characters.

The **probability of an *n*-gram**, $c_1, ..., c_n$, appearing in a text is:

$$\begin{aligned} P(c_1, ..., c_n) &= \prod_{i=1}^{n} P(c_i \mid c_1, ..., c_{i-1}) \\ &= P(c_1)P(c_2|c_1)P(c_3|c_1, c_2)...P(c_n|c_1, ..., c_{n-1}) \end{aligned}$$

An ***n*-gram Character Model** is a probability distribution of $n$-character sequences using a Markov chain of order $n - 1$.

We may model 3-grams with a Markov chain of order 2. Prob. of observing a character in a sequence only depends on the two previous characters.

**Language Identification Problem**: identify the language that a text (corpus) is written in

**An approach to solve the language identification problem**

- Build a 3-gram model for each candidate language $L$, i.e. a probability $P_L(c_i|c_{i-2}, c_{i-1}, L)$ for each possible 3-gram.
- Then compute probability $P(L|\text{text})$ for each $L$ and select the most probable language $L^*$ for the text with Bayes rule:

$$\begin{aligned} L^* &= \arg\max_L \{P(L|c_1, ..., c_n)\} \\ &= \arg\max_L \{P(L) \cdot P(c_1, ..., c_n|L)\} \\ &= \arg\max_L \left\{ \Pr(L) \cdot \prod_{i=1}^{n} P(c_i|c_{i-2}, c_{i-1}, L) \right\} \end{aligned}$$

Trigram/3-gram models can also be used for:

- Spelling correction: identify and fix errors in word spelling (also can use context, e.g. to, too, two)
- Genre classification: identify cateogry of text (e.g. news, novel, blog, ...)

We may encounter some **issues with *n*-gram models**:

- Training data (corpus) provides only an estimate of true probability distribution.
- For common character sequences, e.g. “th”, corpus gives a good estimate, e.g. $P(\text{"th"}) = 1.5\%$
  But for uncommon seq. such as “ht”, there are no English dictionary that contains words beginning with this sequence so training could result in $P(\text{"ht"}) = 0$.
- Resulting model may not generalise well to unseen texts, e.g. “http” might be impossible.

**Smoothing** is the process of assigning a non-zero probability to sequences that do not appear in the training corpus.

> **Note**
>
> We can use **Linear Interpolation Smoothing** to combine unigrams, bigrams, and trigrams:
>
> $$\hat{P}(c_i|c_{i-2}, c_{i-1}) = \lambda_1 P(c_i) + \lambda_2 P(c_i|c_{i-1}) + \lambda_3 P(c_i|c_{i-2}, c_{i-1})$$

> **Note**
>
> We can use **cross-validation to build $n$-gram models**:
> - split corpus into training/validation corpus
> - build multiple models with different paramter values, e.g. $n$ and $\lambda$ using validation corpus
> - select the one that performs the best on the training corpus
>
> for language identification, compute accuracy

More on language models:
- **word**: sequence of characters
- **vocabulary**: set of valid words in a language
- **out-of-vocabulary words**: words with characters of the language that are not valid
- **$n$-gram word models**: similar to character models except that they model sequences of words rather than of characters
- We may use an additional word <UNK> (unknown word) in the vocabulary for words in the corpus without meaning

**$n$-gram examples for English**

- unigram: ‘logical | are | as | a | confusion | may | ...“
- bigram: ‘systems are | a very | similar computationally | ...“
- trigram: ‘planning and scheduling | are integrated the | ...“

## 7.2. Text Classification

**Text Classification/Categorisation** is, given a text and predefined classes, deciding the class which some text belongs to, e.g. we could do: topic analysis, sentiment analysis.

**Spam detection example**

Classify an instance as spam or ham.

- Training corpus: mails in inbox folder (hams), in spam folder (spams)
- Spam emails may contain:
  - $n$-grams like 'cheap', 'a bargain', 'you can buy', etc
  - punctuation marks to confuse spam detectors, e.g. 'yo,u d-eserve'

In this example:

- **Instances**: messages, i.e. vectors of feature-value pairs
- **Features**: words or unigrams or terms in the vocabulary
- **Values**: number of occurrences of each word in the message
- We may apply any classification algorithm, e.g. decision trees, naive Bayes, k-nearest neighbours, to our data set

Some enchancements:

- Use $n$-grams with $n > 1$ to not lose word order information.
- Other attributes can be included such as delivery time, sender, and receiver.
- Term-document matrix tends to sparse, we may perform feature selection to identify most appropriate features for distinguishing spam/ham.

**Bayesian Approach**

Compute an $n$-gram model for $P(\text{msg}|\text{spam})$ and $P(\text{msg}|\neg\ \text{spam})$ by training in the spam and ham folder. A new *msg* can then by classified using Bayes rule:

$$\underset{c\in\{\text{spam},\neg\ \text{spam}\}}{\arg\max}\ \Pr(c|\text{msg}) = \underset{c\in\{\text{spam},\neg\ \text{spam}\}}{\arg\max}\ \Pr(\text{msg}|c)\Pr(c)$$

Priors can be set by computing proportions:

$$\Pr(\text{spam}) = \frac{|\text{spam}|}{|\text{spam}| + |\neg\ \text{spam}|}$$

## 7.3. Information Extraction

**Information extraction** is the process of acquiring knowledge by skimming texts and looking for special types of objects as well as relationships between them.

It allows finding entites and storing them in a database, e.g. extract addresses from webpages and store their fields (street, postcode, city), extract instances of storms from weather reports and store their fields (temp., humidity, wind speed).

Usually information extraction refers to following closely related tasks:
- **Named Entity Recognition**: identify named entities such as locations, people, and organisations
- **Relationship Extraction**: identify relationships between entities

**A pipeline for information extraction**
1. **Tokenisation**: segment text into tokens, i.e. individual units of meaning
2. **Complex Words**: recognise complex words such as collocations ('set up', 'joint venture') or names ('Isle of Man') appearing as combinations of lexical entires.
   e.g. a company name can use the regular expression: Capitalised Word + ("Co" | "Inc" | "Ltd")
3. **Basic Groups**: chunk text into units such as:
   NG (noun phrase/group), VG (verb phrase/group), PR (preposition)
   e.g. an example text may obtain the following groups:

| | |
|---|---|
| NG | Monsters Inc |
| VG | said |
| NG | Friday |
| NG | it |
| VG | had set up |
| NG | a joint venture |
| PR | on |
| NG | the Isle of Man |

4. **Complex-phrases**: define rules for describing complex relationships
   e.g. Company + SetUpJointVenture((Company)*)
5. **Structure merging**: merges terms referring to the same entity
   e.g. 'joint venture' refers to the same entity in sentences 'has set up a joint venture' and 'the joint venture will'

### 7.3.1. Regular Expressions

A **regular expression**, $R$, defines a search pattern and is associated with a set of $L(R)$ of strings.

- **Alphabet** $\Sigma$: finite set of symbols
- **String** $s = c_1, c_2, ..., c_n$: sequence with characters of $\Sigma$
- $\Sigma^*$: set of all strings (of any length) that can be built from $\Sigma$
- $\Sigma^n$: set of strings of length $n$ that can be obtained from $\Sigma$
- $L \subseteq \Sigma^*$: subset of strings obtained with the alphabet

Regular expressions are appropriate for consistent domains with well-specified formats that enable using relatively easily defined templates.

### 7.3.2. Hidden Markov Models

A major challenge in probablistic modelling is *dimensionality*: as the no. of dimensions/attributes grows, constructing models becomes rapidly challenging. Model complexity tends to grow exponentially with dimension.

**Factorisation** of a probability distribution function into smaller components allows us to compute simpler models of multi-variate data.

**Factorisation using independence**

Joint probability distribution of $n$ independent variables:

$$P(x_1, x_2, ..., x_n) = \prod_{j=1}^{n} P(x_j)$$

Where $P(x_j)$ is a one-dimensional probability distribution for $x_j$. Independence allows factorisation of a density function into simpler component parts and provides a technique for constructing simple models from multi-variate data.

Although the independence assumption is an approximation, it is useful for many applications.
In some practical situations, it is not a valid assumption and dependence has to be explicit.

**Factorisation into a sequence of conditional distributions**

Joint probability distribution of $n$ potentially dependent variables:

$$P(x_1, ..., x_n) = P(x_1) \prod_{j=2}^{n} P(x_j | x_1, x_2, ..., x_{j-1})$$

The product involves a sequence of conditional distributions. Determining $P(x_1, ..., x_m)$ is expensive because it requires computing an exponential number of values to the number of variables (assuming that $x_j$'s take discrete values). This is a fully specified density model.

**First-Order Markov Model**

Dependencies can be modeled as a graphical model, i.e. directed graph with nodes corresponding to variables and arcs to actual dependencies. Often, each $x_i$ depends on only few predecessors which we exploit to derive models that generalise better.

$$P(\boldsymbol{x}_j | \boldsymbol{x}_{j-1}, ..., \boldsymbol{x}_1) = P(\boldsymbol{x}_j | \boldsymbol{x}_{j-1})$$


e.g. a 2-gram character model

**Markov assumption**: when predicting the future, the past doesn't matter, only the present.
All information relevant to $x_j$ is contained in the immediately precdeing variables $x_{j-1}$.
Nodes correspond to variables and arcs indicate dependencies between variables.

***k*-th Order Markov Models**

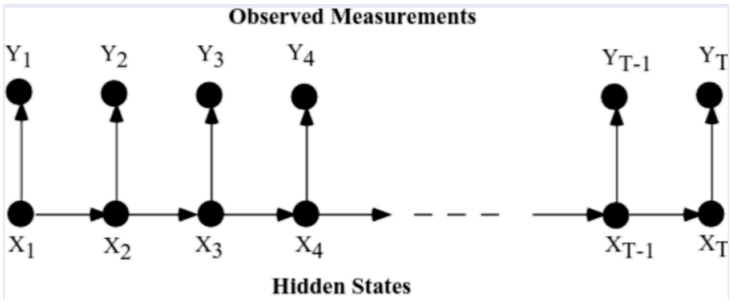
$x_i$ may depend on earlier measurements in the sequence and not just the previous $x_{i-1}$:

$$P(x_1, ..., x_n) = P(x_1) \prod_{j=1}^{n} P(x_j | x_{j-1}, ..., x_{j-k})$$

Often, first order Markov models are not accurate, e.g. text sequences where $x_j$ takes a value such as a verb, adjective, noun, etc.

A **Hidden Markov Model** is a generalisation of Markov models including hidden variables.

- $y_1, ..., y_n$: observations in a sequence, e.g. words
- $x_1, ..., x_n$: hidden states in a sequence, e.g. tags <location>, <person>, <other>
- Observation $y_j$ depends only on the state $x_j$
- Hidden state $x_j$ depends only on the previous state $x_{j-1}$ and may also depend on the previous word $y_{j-1}$.




The joint distribution of observed sequence $y_1, ..., y_n$ and hidden state sequence $x_1, ..., x_n$:

$$P(y_1, ..., y_n, x_1, ..., x_n) = P(x_1)P(y_1|x_1)\prod_{j=2}^{n} P(y_j|x_j)P(x_j|x_{j-1})$$

Only $y_j$'s are observed and $x_j$'s are hidden, i.e. there is uncertainty of generation of observations.

- $x_j$ is sampled from probability distribution $P(x_j|x_{j-1})$
- $y_j$ is samples from $P(y_j|x_j)$

Using a training corpus for which the sequences of hidden states are known, we may extract information from new texts.

## 7.4. Information Retrieval

### 7.4.1. Text Representation

Text representation is the task of finding texts relevant to a user's query.

- Corpus of documents: E.g. paragraphs, webpages, articles, posts, emails.
- Query: Information requested by the user.
- Result Set: Subset of documents matching the user's query.

Major information retrieval approaches are based on **word count statistics**.

**Term-Document Matrix** is a table with the frequencies of words in a collection of documents. Rows correspond to documents. Columns represent terms.

Example term-document matrix

| | **I** | **like** | **dislike** | **data** | **mining** |
|---|---|---|---|---|---|
| $\boldsymbol{D_1}$ | 1 | 2 | 0 | 1 | 1 |
| $\boldsymbol{D_2}$ | 1 | 0 | 1 | 1 | 1 |

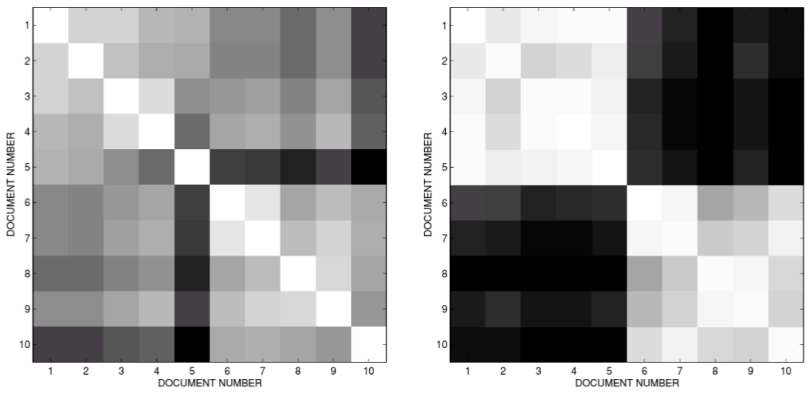
Each document is represented by a vector $\boldsymbol{u} = (u_1, u_2, ..., u_n)$ where $u_j$ is the weight of the term $t_j$ in document $\boldsymbol{u}$. We could either use a boolean representation ($u_j$ is 1 if it appears in $D_i$ or 0 otherwise) or a vector space representation where $u_j$ is a real-valued number (e.g. frequency).

The **cosine distance** of two vectors $\boldsymbol{u} = (u_1, ..., u_n)$ and $\boldsymbol{v} = (v_1, ..., v_n)$ is:

$$d(\boldsymbol{u}, \boldsymbol{v}) = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}}$$

Example distances of term-document matrix

| | $\boldsymbol{t_1}$ | $\boldsymbol{t_2}$ | $\boldsymbol{t_3}$ | $\boldsymbol{t_4}$ | $\boldsymbol{t_5}$ | $\boldsymbol{t_6}$ |
|---|---|---|---|---|---|---|
| $\boldsymbol{d_1}$ | 24 | 21 | 9 | 0 | 0 | 3 |
| $\boldsymbol{d_2}$ | 32 | 10 | 5 | 0 | 3 | 0 |
| $\boldsymbol{d_3}$ | 12 | 16 | 6 | 0 | 0 | 0 |
| $\boldsymbol{d_4}$ | 6 | 7 | 2 | 0 | 0 | 0 |
| $\boldsymbol{d_5}$ | 43 | 31 | 20 | 0 | 3 | 0 |
| $\boldsymbol{d_6}$ | 2 | 0 | 0 | 18 | 7 | 16 |
| $\boldsymbol{d_7}$ | 0 | 0 | 1 | 32 | 12 | 0 |
| $\boldsymbol{d_8}$ | 3 | 0 | 0 | 22 | 4 | 2 |
| $\boldsymbol{d_9}$ | 1 | 0 | 0 | 34 | 27 | 25 |
| $\boldsymbol{d_{10}}$ | 6 | 0 | 0 | 17 | 4 | 23 |




(a) Euclidean distance (b) cosine distance

### 7.4.2. Methods

In this section:
- we are interested in less precise queries
- queries are based on similarity rather than equality or inequality
- methods tend to be interactive with user feedback about relevance to score retrieval process

**Retrieval by Content**

Find $k$ objects that are most similar to an object query:
- Queries can be represented as **term vectors**: $\boldsymbol{q} = (q_1, q_2, ..., q_n)$ similar to documents
- Term vector contains weights based on relative importance of terms:
  - Terms not in query get weight 0
  - Terms in the query are typically weighted between 0 and 1
- We can find the $k$ closest documents by computing the cosine distances $d(\boldsymbol{q}, \boldsymbol{u}_i)$

**Enhancements**
- **Case folding**: convert terms to lowercase
- **Stemming**: reduce all words to their stem
- **Stop word removal**: e.g. remove articles the, a/an and prepositions in, at, on, of, to
- **Non-alphabetic character removal**: e.g. punctuation marks and numbers
- **Word embeddings**: use similar representations for words with similar meanings
- **Metadata parsing**: exploit data outside the documents

**Latent Semantic Indexing (LSI)**
- Previous approaches ignore the semantics.
  LSI assumes existence of hidden semantic structure in documents.
- LSI performs **Principal Components Analysis (PCA)** on the $m \times n$ document-term matrix to compute an $m \times k$ matrix of lower dimension, i.e. $k \ll n$.
- Computes $k$ principal component diections in the $m \times n$ space.
  Intuitively, terms that appear frequently together are combined to form a single principal component.

**PageRank Algorithm**
- Google's algorithm for web search which identifies pages relevant to a query, but also ranks them.
- Pages with many in-links are ranked higher. Links from high-ranked pages are weighted better.

$$R(p) = \frac{1-d}{n} + d \sum_{i=1}^{n} \left( \frac{R(\text{in}_i)}{C(\text{in}_i)} \right)$$

### 7.4.3. Evaluation

Evaluating information retrieval systems requires assessing relevance, i.e. whether the information retrieved is useful to the user that invoked the query. Measuring relevance is challenging because the degree of usefulness differs (i) from user to user, (ii) from task to task, and (iii) over time for the same user and same task.

We can use:
- Precision, Recall, F1-Score
- Precision-Recall Performance Curves

**TF-IDF Score**: a document gets high **Term Frequency (TF)** score if the frequency of a term is high in the document. Words appearing in many documents (e.g. "and") take high TF but low **Inverse Document Frequency (IDF)** score. TF-IDF highlight rare query items.

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

F1-Score is the harmonic mean of precision and recall.

# 8. Association Rules

An **association rule** is an expression in if-then format, the if-part is the precondition/antecedent and consists of a series of tests while the then part is the conclusion/consequent and assigns values to one or more attributes.

Association rules can be computed with a bottom-up generate-and-test procedure, successively generating expressions that frequently occur in the rule's right-hand side.
Association rules may incorporate any attribute or combination of attributes as the target.
We are specifically interested in rules that apply to reasonably large no. of instances w/ high acc.

We want the following metrics to be high:
- **Coverage/support count**: no. of instances covered by rule
- **Support**: proportion of instances covered by rule
- **Confidence/accuracy**: proportion of instances that the rule predicts correctly over all instances

An **item** is an (attribute, value) pair.
A **k-item set** is a combination of $k$ different (attribute, value) pairs with prespecified min. coverage.
Association rules are derived from item sets.

> **Note**
>
> Applications in:
> - market basket analysis: use data about customer purchases and provide insights about what customers buy and when
> - bundle services: determine service bundles for telecommunication customers, e.g. phone/internet, to maximise revenue
> - fraud detection: identify signs of fraud by finding unusual combinations of insurance claims
> - medical treatment: indications of likely complications due to treatment combinations

**Example**

Consider the following dataset:

| Customer | Items |
|---|---|
| 1 | Orange juice, soda |
| 2 | Milk, orange juice, window cleaner |
| 3 | Orange juice, detergent |
| 4 | Orange juice, detergent, soda |
| 5 | Window cleaner, soda |

And selecting the association rule, orange juice $\longrightarrow$ detergent:
- Coverage/support count is 2
- Support is $\frac{2}{5} = 40\%$
- Confidence/accuracy is $\frac{2}{4} = 50\%$

## 8.1. Apriori Algorithm

**Algorithm 1:** Apriori Algorithm for generating association rules

min-coverage, min-confidence ← some predefined values
item-sets ← 1-item sets of all items which satisfy min-coverage
**while** $i = 2, \ldots, \infty$-item set is possible **do**
  item-sets ← all $i$-item sets made from pairs of item-sets that satisfy min-coverage
**for** item-set **in** item-sets **do**
  produce association rules to split relevant items between precondition and conclusion
**for** generated-rule **do**
  evaluate confidence and retain if rule meets min-confidence

Example using basket data

Assume min-coverage is 2 and min-confidence is 0.6.

| 1-item sets | coverage |
|---|---|
| **orange juice** | **4** |
| **soda** | **3** |
| milk | 1 |
| **window cleaner** | **2** |
| **detergent** | **2** |

| 2-item sets | coverage |
|---|---|
| **orange juice, soda** | **2** |
| orange juice, window cleaner | 1 |
| **orange juice, detergent** | **2** |
| soda, window cleaner | 1 |
| soda, detergent | 1 |
| window cleaner, detergent | 0 |

| 3-item sets | coverage |
|---|---|
| orange juice, soda, detergent | 1 |

No 3-item set meets minimum coverage requirements, thus association rules can only be obtained from the following 2-item sets:

| item sets satisfying min-coverage | coverage |
|---|---|
| orange juice, soda | 2 |
| orange juice, detergent | 2 |

Finally, generate and discard rules that do not satisfy min-confidence:

| rule | confidence |
|---|---|
| OJ $\longrightarrow$ S | $\frac{2}{4} = 0.5$ |
| S $\longrightarrow$ OJ | $\frac{2}{3} = 0.67$ |
| OJ $\longrightarrow$ D | $\frac{2}{4} = 0.5$ |
| D $\longrightarrow$ OJ | $\frac{2}{2} = 1.0$ |

## 8.2. FP Growth Algorithm

**Algorithm 2:** FP Growth Algorithm for learning assocation rules

1 find all $k = 1$ frequent item sets
2 grow item sets from size $k$ to $k + 1$ by pruning item sets containing subsets with support $< T_s$

> pruning might be possible without re-examining the data, e.g. consider $k$-item sets $(A, B)$ and $(B, C)$ for $k = 2$, the extended set $(A, B, C)$ where $k + 1 = 3$: if either $(A, B)$, $(B, C)$ is not frequent, we can prune $(A, B, C)$ otherwise we computes support for $(A, B, C)$ by considering every single instance.

3 stop pruning when $k$ reaches predefined limit
4 convert item sets to association rules and remove any rules with confidence $< T_c$

The FP Growth algorithm leverages the property that, if an item-set is not frequent, then all its supersets are also not frequent. Consider support(milk) = 1 from earlier example, no item-set with milk is above the coverage threshold.

To find all item-sets meeting minimum support threshold:
1. start with $k = 1$
2. compute all $k$-item sets and determine their support
3. retain only $k$-item sets satisfying the support threshold
4. using these $k$-item sets, generate $(k + 1)$-item sets for all possible combinations and repeat

> **Note**
>
> FP Growth algorithm follows a generate-and-test methodology:
> - successively generate longer candidate item sets from shorter ones known to be frequent
> - each new candidate itemset requires a scan of the data set (may be costly)
> - we can make this more efficient using data structures such as an FP-tree

### FP-Growth Algorithm

Count each item frequency and sort in descending order. Perform a second pass of the data to create the FP-tree (a compressed representation of input data). Recursively process the tree to identify frequent item sets.

We only perform two passes on the training data, so individual items not meeting min. confidence threshold are not inserted into the tree.

**Table 6.1** Preparing Weather Data for Insertion into an FP-Tree

| (a) | Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|---|
| 1 | sunny | hot | high | false | no |
| 2 | sunny | hot | high | true | no |
| 3 | overcast | hot | high | false | yes |
| 4 | rainy | mild | high | false | yes |
| 5 | rainy | cool | normal | false | yes |
| 6 | rainy | cool | normal | true | no |
| 7 | overcast | cool | normal | true | yes |
| 8 | sunny | mild | high | false | no |
| 9 | sunny | cool | normal | false | yes |
| 10 | rainy | mild | normal | false | yes |
| 11 | sunny | mild | normal | true | yes |
| 12 | overcast | mild | high | true | yes |
| 13 | overcast | hot | normal | false | yes |
| 14 | rainy | mild | high | true | no |

| (b) | |
|---|---|
| **play = yes** | **9** |
| **windy = false** | **8** |
| **humidity = normal** | **7** |
| **humidity = high** | **7** |
| **windy = true** | **6** |
| **temperature = mild** | **6** |
| play = no | 5 |
| outlook = sunny | 5 |
| outlook = rainy | 5 |
| temperature = hot | 4 |
| temperature = cool | 4 |
| outlook = overcast | 4 |

**Table 6.1** Preparing Weather Data for Insertion into an FP-Tree *Continued*

| (c) | |
|---|---|
| 1 | **windy = false, humidity = high,** play = no, outlook = sunny, temperature = hot |
| 2 | **humidity = high, windy = true,** play = no, outlook = sunny, temperature = hot |
| 3 | **play = yes, windy = false, humidity = high,** temperature = hot, outlook = overcast |
| 4 | **play = yes, windy = false, humidity = high, temperature = mild,** outlook = rainy |
| 5 | **play = yes, windy = false, humidity = normal,** outlook = rainy, temperature = cool |
| 6 | **humidity = normal, windy = true,** play = no, outlook = rainy, temperature = cool |
| 7 | **play = yes, humidity = normal, windy = true,** temperature = cool, outlook = overcast |
| 8 | **windy = false, humidity = high, temperature = mild,** play = no, outlook = sunny |
| 9 | **play = yes, windy = false, humidity = normal,** outlook = sunny, temperature = cool |
| 10 | **play = yes, windy = false, humidity = normal, temperature = mild,** outlook = rainy |
| 11 | **play = yes, humidity = normal, windy = true, temperature = mild,** outlook = sunny |
| 12 | **play = yes, humidity = high, windy = true, temperature = mild,** outlook = overcast |
| 13 | **play = yes, windy = false, humidity = normal,** temperature = hot, outlook = overcast |
| 14 | **humidity = high, windy = true, temperature = mild,** play = no, outlook = rainy |

| (d) | |
|---|---|
| play = yes and windy = false | 6 |
| play = yes and humidity = normal | 6 |

*(a) The original data, (b) frequency ordering of items with frequent item sets in bold, (c) the data with each instance sorted into frequency order, and (d) the two multiple-item frequent item sets.*

Figure 73: Sample data to be inserted into FP-tree

The header table on left-hand side shows frequencies of individual items satisfying min. coverage threshold sorted in descending order.

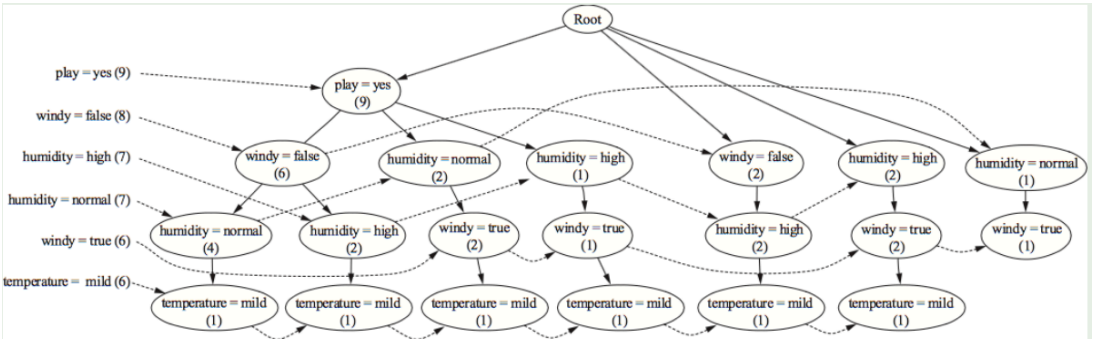



Figure 74: Resulting FP-tree

The *FP-tree* is shown with solid arrows. A node number indicates how many times the sorted order of items (incl. node's item) occur in the data set. Dashed arrows allow efficiently deriving frequent $k$-item sets apart from ones that are already in tree. Item sets can be computed starting from bottom of header table traversing through dashed arrows. After completing with item sets of last element, we proceed upwards. In this way, we can detect frequent item sets.

> **Note**
>
> In this case, only two item sets satisfy minimum support threshold:
>
> $$\text{play} = \text{yes}$$
> $$(\text{play} = \text{yes}) \cap (\text{windy} = \text{false})$$

## 8.3. Evaluating Association Rules

Measures for evaluating quality of association rules:

1. Coverage
2. Support
3. Confidence
4. Lift
5. Contigency Table

> **Note**
>
> Say rule $S \to OJ$ has confidence $66.7\%$ however 4 out of 5 transactions contain OJ.
> High confidence suggests strong association rule but it can be misleading.
> If the consequent has high support, then we may have high confidence even if antecedent and consequent are independent!

**Lift**: measure the usefulness of a rule by comparing with respect to random guessing.
Compare confidence of rule to confidence of **null rule** which has same RHS but empty LHS.

**Contigency Table**

A **contigency table** has two dimensions:

- Does antecedent/LHS of rule appear in transaction?
- Does consequent/RHS of rule appear in transaction?
- Number of transactions is equal to sum of the table.
- If the *bottom right cell is higher than other cells*, then we have a good indication rule is good.

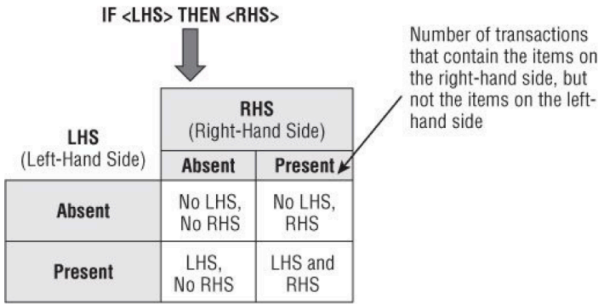



Figure 75: Contigency Table of an Association Rule

**Possible characteristics of assocation rules**

- **Actionable**: can be justified by telling a story, possible actions can be determined
  "Walmart customers who purchase Barbie [..] have 60% likelihood of purchasing [..] candy bars"
  Actions are better product placement and advertisements, product promotions.
- **Inexplicable**: seem to have no explanation and do not suggest a course of action; may trigger further analysis, e.g. to identify an anomaly
  "When a new hardware store opens, one of the most commonly sold items is toilet bowl cleaners"
  Are they easier to find at store opening and not other times? Larger discount than other products?
- **Trivial**: generate in a valid way, well-supported by data, but corresponding to well-known facts that do not provide valuable information
  "Customers who purchase maintenance agreements are likely to purchase large applicables"
  Why else would they but maintenance agreements? Others: hamburgers/buns, paints/brushes, etc.

## 8.4. Extensions

### 8.4.1. Sequence Analysis

**Sequence Analysis/Sequential Pattern Analysis** adds the element of time to association analysis. We consider sequences, not just associations, of items, where the order in time is important.

> **Note**
>
> Applications:
> - shopping sequences: e.g. new homeowners purchase shower curtains before furniture
> - customer behaviour: e.g. when customers visit their bank for a statement of their accounts, this is a sign they might transfer to a competitor within two weeks
> - clickstream analysis for website visits

Association analysis techniques can be extended to sequence analysis, but require accounting for order of items: support of an item set not depends on the order of the items, e.g. the support of Juice, Soda ⟶ Milk is 1 not 2.

**Generalised sequential pattern (GSP)** is a suitable algorithm for sequence mining based on the Apriori algorithm.

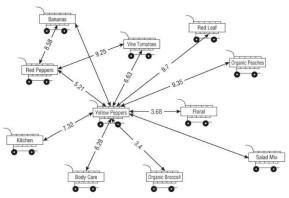
Example data

| Customer | Purchase Seq. |
|---|---|
| 1 | juice, soda, juice, milk, detergent |
| 2 | juice, milk, milk, detergent |
| 3 | milk, juice, soda |

### 8.4.2. Link Analysis

**Link Analysis** is an approach for analysing and exploiting relationships and connections between items, based on graph theory. Represent relationships as items as edges in a graph. Can be used for supervised and unsupervised learning. Used for social net. analysis, crime/fraud detect, marketing.

In context:
- nodes may correspond to items in transactions
- edges may represent items appearing in same transaction or other item relationships
- weights could specify fraction of transactions that each pair of items appears to



Graphs could be undirected/directed based on if direction of link is important, multigraph when multiple edges between same pair of items is allowed, connected if there is a path between every pair on nodes, i.e. consecutive edges going from one node to another.

Analysing transactions with graphs may require different computations such as:
- **Degree centrality**: evaluating relative importance of a node to the graph
- **Closeness centrality**: measuring how close a node is to all other nodes in the graph

# 9. Temporal Data Mining

A **time series** is a set of values obtained from measurements over time, e.g. rainfall, stock market prices, or exchange rates measured on a daily basis.

Formally, a **time series** is an ordered sequence, $T = (x_1, ..., x_n)$ where $x_t$ is a time series value at time $t$. When plotting, horizontal axis is usually time and vertical axis corresponds to values of $x_t$.

- A data set may contain a single or multiple time series.
- Successive observations are not independent, as they rely on past and present observations which influence future ones.

| | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|
| 1960 | 0.71 | 0.63 | 0.85 | 0.44 |
| 1961 | 0.61 | 0.69 | 0.92 | 0.55 |
| 1962 | 0.72 | 0.77 | 0.92 | 0.60 |
| 1963 | 0.83 | 0.80 | 1.00 | 0.77 |
| 1964 | 0.92 | 1.00 | 1.24 | 1.00 |
| 1965 | 1.16 | 1.30 | 1.45 | 1.25 |
| 1966 | 1.26 | 1.38 | 1.86 | 1.56 |
| … | … | … | … | … |
| 1974 | 6.03 | 6.39 | 6.93 | 5.85 |
| 1975 | 6.93 | 7.74 | 7.83 | 6.12 |
| 1976 | 7.74 | 8.91 | 8.28 | 6.84 |
| 1977 | 9.54 | 10.26 | 9.54 | 8.73 |
| 1978 | 11.88 | 12.06 | 12.15 | 8.91 |
| 1979 | 14.04 | 12.96 | 14.85 | 9.99 |
| 1980 | 16.20 | 14.67 | 16.02 | 11.61 |

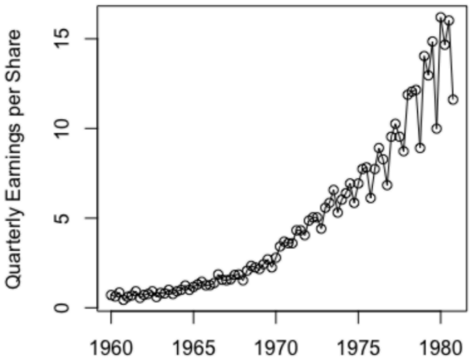



Figure 77: J&J Quarterly Earnings per Share

## 9.1. Time Series Analysis

Traditional approaches **decompose** a time series into additive components as such:

$$x_t = T_t + C_t + S_t + \varepsilon_t$$

- Trend $T_t$: variations of low frequency
- Seasonality $S_t$: regular predictable changes
- Cycle $C_t$: periodic variations
- Error $\varepsilon_t$: captures uncertainty of the model

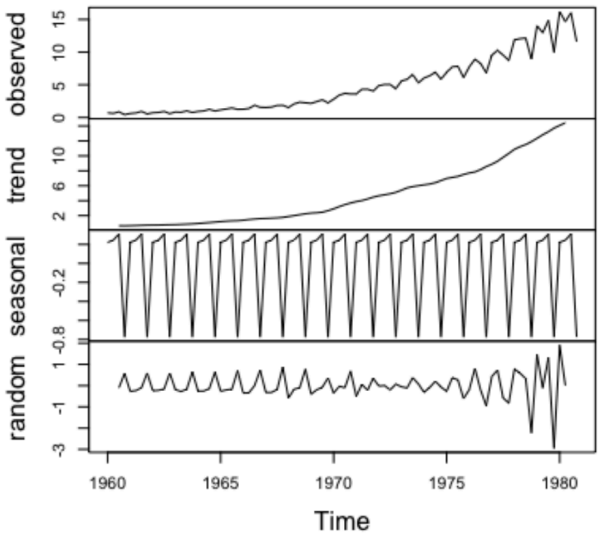



Figure 78: Decomposition of additive time series

**Basic approaches for modelling time series**

- **Moving Average (MA)**

$$\hat{x}_{t+1} = \frac{x_{t-k+1} + ... + x_{t-1} + x_t}{k}$$

- **AutoRegressive (AR)**

$$|(x)_{t+1} = \beta_0 + \beta_1 x_t + \varepsilon_t$$

- **Exponential Smoothing**

$$\hat{x}_1 = x_1$$

$$\hat{x}_{t+1} = \alpha \cdot x_t + (1 - \alpha) \cdot \hat{x}_t \text{ for } t \geq 1$$

$\hat{x}_t$ : estimate of $x_t$ computed as weighted average of past observations

$\alpha$ : smoothing factor

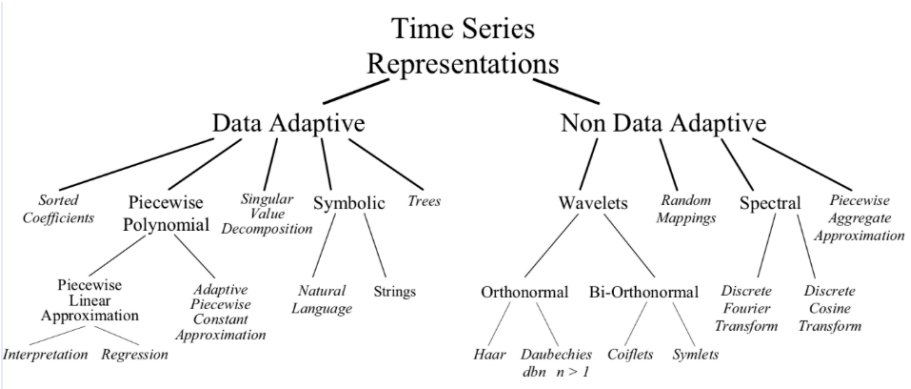
**Challenges with time series data**

- **Representation** of time series data in a way that the major characteristics are retained while dimensionality is reduced
- **Similarity**: measuring distance between pairs of time series so as to distinguish or match them
- **Indexing**: organising massive sets of time series so as to achieve low memory requirements and enable fast querying

**Representation of time series data**

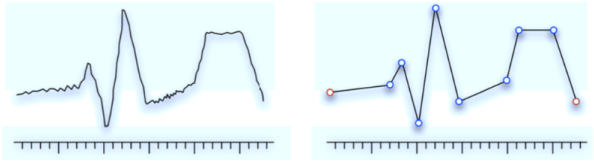
Typically, representations reduce dimensionality.

- **Non-Adaptive**: common representation for all parts of a time series
  - **Piecewise Linear Approximation**: use piecewise linear segments to approx. time series
  - **Single Value Decomposition (SVD)**: represent the shape in terms of linear combination of basic shapes, based on variance of the data set; in constrast to DFT which is local, SVD is global
- **Adaptive**: local properties are used to construct non-uniform representations of a time series
  - **Discrete Fourier Transformation (DFT)**: any signal can be represented with finite number of sine/cosine waves




**Segmentation**

Given a time series $T$ with a large number $n$ of time points, compute a model with $k < n$ peicewise segments approximating $T$:



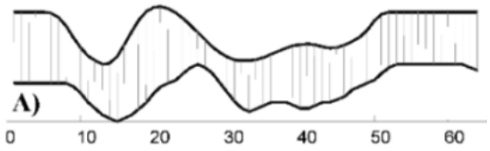
- Pre-processing step as prep. for data mining.
- Trend analysis technique.
- Can be accomplished with simple time discretisation which partitions using fixed-length window.

**Measuring Similarity**

- *Whole series matching*: compare two time series $T$ and $T'$ using their entire length, e.g. find stocks behaving similarly
- *Subsequence matching*: given large sequence $T$ and query subsequence $Q$, find the $T$'s subsequences matching $Q$, e.g. significant stock price variations

**Euclidean Distance**

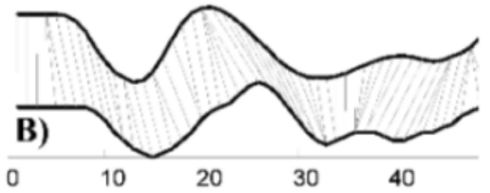
Given two time series $T$ and $T'$ of the same length, compute distance between the points.




- Standarisation is useful when two time series have different offsets in time axis.
- Can be unsuitable metric because it does not allow acceleration and deceleration along time axis.

**Dynamic Time Warping**

Wrap the time axis of one or both sequences for better alignment.

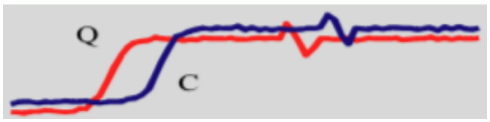



- Time series can be extended by repeating elements.
- Can be implemented using dynamic programming, but time consuming for large-scale applic.

**Longest Common Subsequences**

Match two sequences with potentially unmatched elements:

- $T = \{x_1(T), ..., x_n(T)\}$: time series with $n$ elements
- $T' = \{x_1(T'), ..., x_{n'}(T')\}$: time series with $n'$ elements




- A subsequence is a sequence whose elements appear in same relative order, but not necessarily continuously, in the two original sequences.
- Example: for $T = \{1, 2, 3, 4, 5, 1, 7\}, T' = \{2, 4, 5, 3, 1, 8\}$, longest common subseq. is $\{2, 4, 5, 1\}$.

**Indexing**

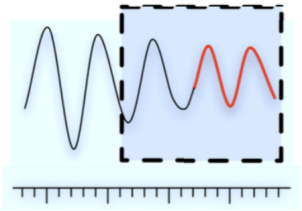
Given a time series $T$ and similarity measure $D(T, T')$, find the most similar time series to $T$ among a set of different time series.

- Simplest approach is a **linear scan** but it can be costly. More efficient implementations initially compute bounds of distances using compressed versions of the time series data.
- Using index structures, that cluster similar time series in same group, we have faster access to the most relevant ones. Efficiency of indexing depends on precision of the approximation in the reduced dimensionality space.

### 9.1.1. Time Series Mining Tasks

**Forecasting**

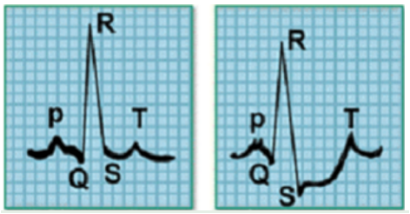
Make predictions about the future based on historical data.
Given a time series $T = (x_1, ..., x_n)$, predict $(x_{n+1}, ..., x_{n+k})$



e.g. forecasting electricity demand, market prices, and weather temperature
usually involves solving a regression problem, e.g. linear or NN

**Classification**

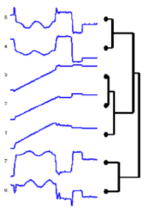
Given an unlabelled time series $T$, assign to $T$ one among a set of predefined discrete labels.




e.g. heartbeat electrical activity can be normal/abnormal
involves exploiting general patterns, local properties, and the nature of the data

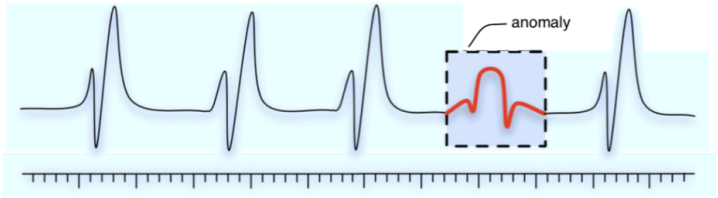
**Clustering**

Given a set of different time series, group similar tiem series in the same cluster.
Or, given a single long time series, **subsequence clustering** performs clustering with smaller time series extracted from the long one with a sliding window.



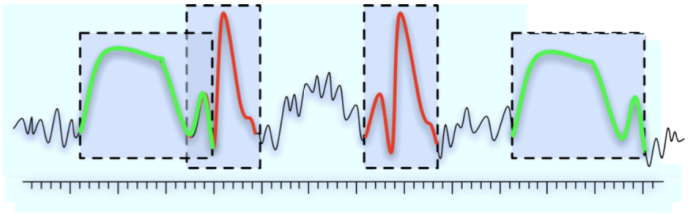
**Anomaly Detection**

Given a time series and model of its behaviour, identify subsequences with anomalies (do not fit).




e.g. surveillance
requires a predictive model first; form of outlier detection

**Motif Detection**
Find every subsequence, **motif**, that appears repeatedly in a longer time series.



inspired from gene analysis in bioinformatics
complex task as several motifs may exist in a single time series, can be of variable length and overlap

## 9.2. Survival Analysis

**Survival Analysis/Time-to-Event Analysis** investigates when something important will happen.

> **Note**
>
> In the context of business and marketing:
> - When a customer is likely to leave; **churn**: no. of customers who stop using a service of a company to move to another company that offers better services and prices for them
> - Next time a customer is likely to return
> - Factor influencing customer loyalty; **tenure**: length of time an individual remains the customer of a company
>
> Applications in:
> - customer analysis/marketing (customer, tenure)
> - clinical data analysis (patient, survival)
> - failure analysis in manufacturing (machine, failure)

A **Survival Curve** is a cumulative histogram, starting from 100% and continuously descending, not necessarily to zero. Stopping is a one-time event, once it happens the customer is not reincluded in analysis.

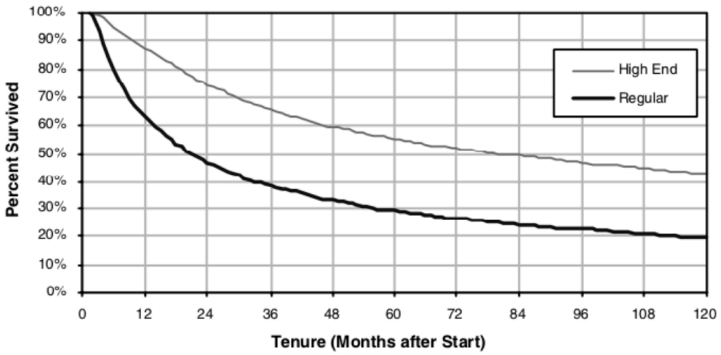



Figure 88: A plot of tenure
Comparing two groups of customers beginning at same time, for a 10 year period

The **customer half-life/median customer lifetime point** is the point at which 50% of customers have been lost, useful measure as it is not affected by customers with very long/short tenures. The **average customer lifetime** can be approximated by area under curve, allows computing average customer worth.

The **survival function** is the probability $S(t)$ that the subject of interest will survive for a period of length $T$ longer than $t$:

$$S(t) = \Pr(T > t)$$

Where $T$: tenure/survival time is a random variable.

> **Note**
>
> Computing tenure requires for each customer: start/stop time
> Survival time data can be continuous or discrete.

**Discrete Survival Time Data**
Time is partitioned in continuous non-overlapping time intervals specified by time points $t_0 = 0 < t_1 < ... < t_\ell$ where time intervals are $(t_0, t_1], (t_1, t_2], ..., (t_{\ell-1}, t_\ell]$

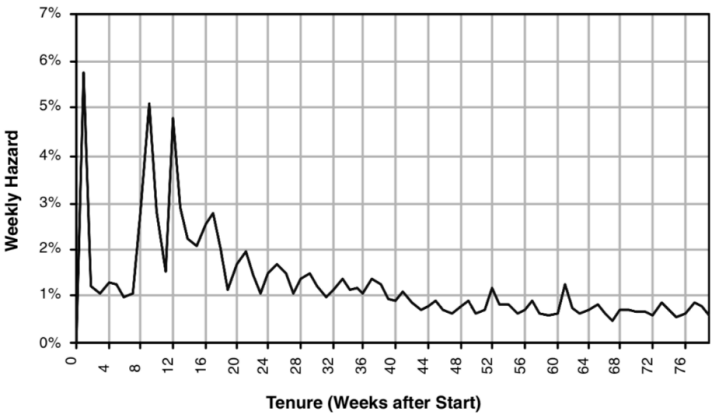
Discrete data frequently arises because time scales are often inherently discrete and events are typically observed/stored at discrete time points.

A survival time is **censored** if it belongs to a time interval where the exact time is unknown.
A form of the missing value problem, censored times are dropped from calculations in this interval.

**Hazard for Discrete Survival Time Data**
The probability $h_i$ that the subject of interest will fail during $(t_{i-1}, t_i]$ assuming survival at least until $t_{i-1}$:

$$h_i = \Pr(t_{i-1} < T \leq t_i | T > t_{i-1})$$

i.e. risk of losing customers in the interval given; can be extended to continuous survival time data




**Survival** is the probability of surviving until a specific tenure. Suitable for comparing different customer groups as it is cumulative.
**Hazard** is the probability that the customer stops at a particular tenure. Because they identify patterns at specific time points, they make specific causes more apparent.

> **Note**
>
> There exist parametric models assuming an explicit functional form for the survival/hazard.
> e.g. exponential or Gamma distributions

The **Kaplan-Meier Estimator of Survival Function** is given by:

$$\hat{S}(t) = \prod_{i:t_i \leq t} \left(1 - \frac{d_i}{n_i}\right)$$

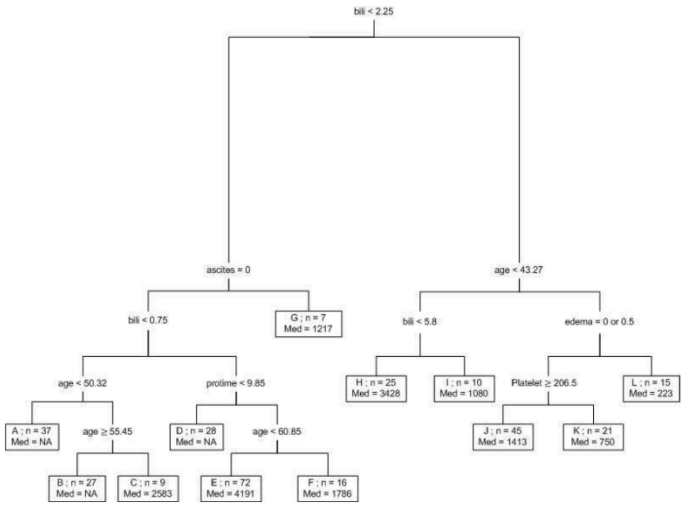
Where $t_1 < t_2 < \ldots < t_\ell$: survival times observed in the data set, $d_i$ is the number of subjects stopping at $t_i$, $n_i$ is the number of subjects known to have survived up to $t_i$, e.g. fraction of patients living for a certain time after treatment.

**Forecasting the number of future customers:**

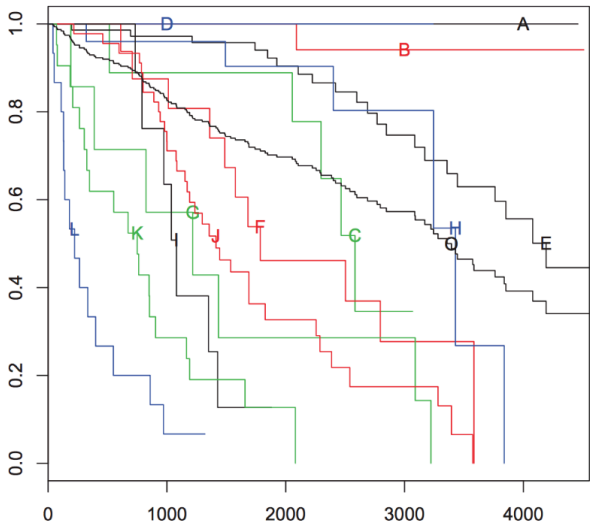
- model exsting customers with various covariates
- use model to forecast when customers stop
- refine survival models to obtain better estimates for new starts and forecast new customer stops

**Survival Trees**

A single tree groups subjects according to their survival behaviour based on covariates.
Different splitting criteria have been proposed.




**Kaplan-Meier survival curves**: each leaf of the tree is associated with an individual curve



# 10. Feature Engineering & Dimensionality Reduction

## 10.1. Derived Variables

- We may find important variables are unused because they are hard to use in current form, we attempt to create **derived variables** representing the same information. It is a hypothesis about a relationship which can be tested in historical data.
- Two variables which are equal in most cases can be informative in the few cases where they are not equal.
- Data from mutliple sources may cover different timeframes. In time series data, use full years to eliminate seasonal effects.
- **Ecological fallacy**: differences between individual members of different groups do not necessarily follow the differences between group summary staistics (e.g. averages).

### 10.1.1. Single-Variable Transformations

**Standardise Numeric Values**

1. Centering: subtract mean, $\mu$, from each value
2. Rescaling: divide each centered value by standard deviation $\sigma$

This process yields a derived variable with $\mu = 0, \sigma = 1$.
This can be useful to provide comparisons between variables using different scales.

e.g. we could convert dollars, percents, days into a common unit
can be particularly useful for clustering/memory-based reasoning wihch depend on distance

**Convert Numeric Values into Percentiles**

Compute percentage of values below each value.
Useful when relative positions is more important than absolute.
Relative measures enable better comparisons.

**Convert Counts into Rates**

Divide by some fixed time unit, useful for counts concerning events over time, e.g. no. of purchases, no. of late deliveries, etc.

**Replace Categorical Variables with Numeric Variables**

Replacing a categorical variable with a numeric one is useful as long as the numbers are meaningful for the data mining task.

Useful approaches to replace categorical variables with numeric:

- numeric descriptor variables: use a handful of numerical variables capturing important aspects of categories based on what needs to be predicted, e.g. city is characterised by population, med. icome, annual rainfall, etc.
- binary indicator variables: create a binary variable for each category where 1 is present, 0 otherwise; works well with few categories

**Replace Numeric Variables with Categorical Variables**

Binning, i.e. discretise similar to histograms
Divide range of numeric variable into a set of subranges (bins) and replace each numeric value

- Equal width binning: bins of equal width, unequal height
- Equal weight binning: divides set of instances into subsets of equal size
- Supervised binning: uses information of target variable to reshape input variable

Useful because it places outliers in one bin

### 10.1.2. Combining Variables

Often data sets contain highly correlated variables.

- Original order value: total value of ordered items
- Net order value: total value of ordered items after subtracting returns and non-delivered out of stock items

- For certain models, e.g. regression models, such variables can be harmful.
  Usually, when two variables are highly-correlated, one should be eliminated.
- Sometimes combining highly correlated variables as ratios can be useful.
- Variables derived by combining highly correlated variables should have **high variance** otherwise they do not add any new information.

**Example: Body Mass Index**

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2}$$

Combined variable may reveal information that is not provided by each variable on its own.

**Example: Price-to-Earnings ratio**

$$\frac{P}{E} = \frac{\text{price per share}}{\text{earnings per share}}$$

A high P/E could mean company's stock is overvalued.

### 10.1.3. Extracting Features from Structured Data

Extracting features from time series data:

- Trend: e.g. no. of tourists might be increasing over the years
- Seasonality: e.g. peak during spring and summer
  To eliminate seasonal effects, we could use complete years or compare specific months/quarters.
- Adding a categorical (or numerical) variable indicating season can be useful to the model.

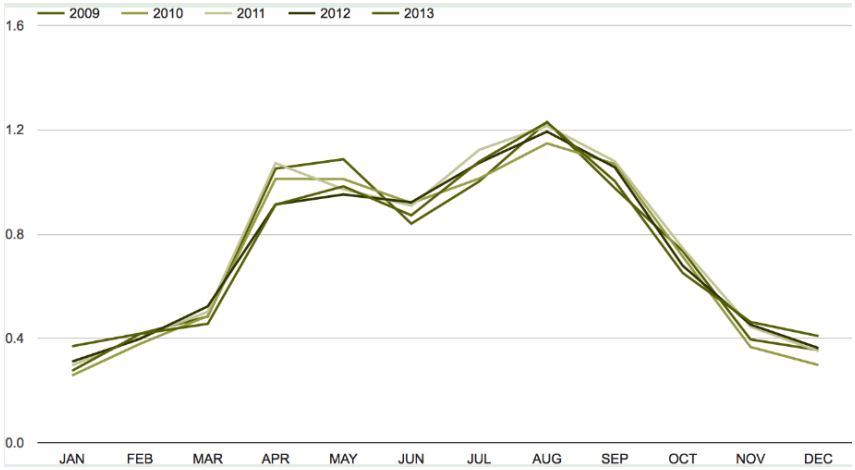
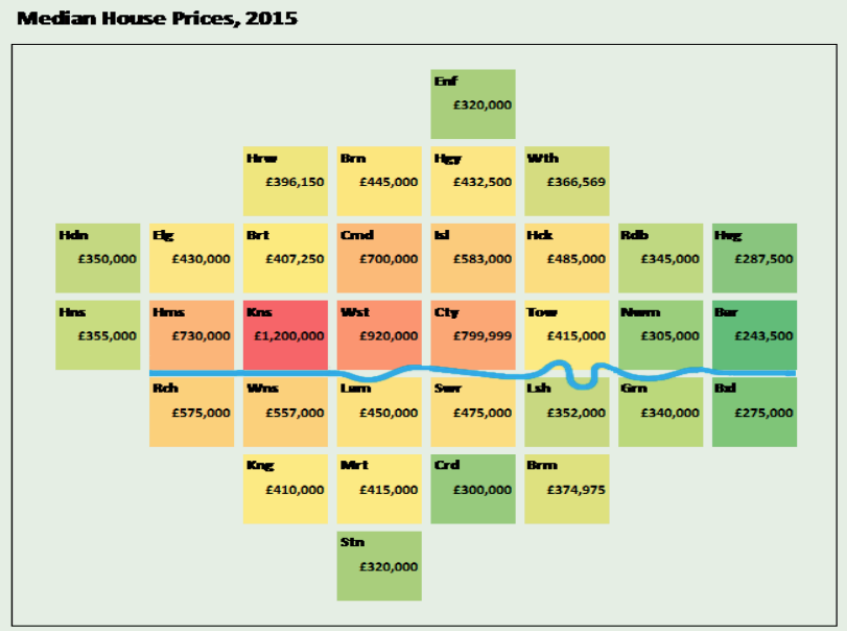



Figure 92: Total number of visitors spending over 3 hours in Peak District National Park.

Extracting features from geographic data:

- Geocoding: translates addresses to actual Earth coordinates
- Mapping: places objects on a map
- Use data, e.g. pop., med. income, temperature, relative to location




### 10.1.4. Handling Sparse Data

**Example: Bank**

A bank offers 30-40 consumer products. For each customer, several attributes are stored with respect to each product, e.g. balance, tenure, etc

This is a sparse data set because each customer only uses a few products.

To handle sparse data:

- Create a small number of dense variables to summarise information from many sparse variables, e.g. no. of accounts, total balance across all accounts.
- Capture patterns across multiple variables in one variable, e.g. categorical variable specifying which types of accounts a customer has.
- Bin together values with populated variables, e.g. reduce no. of possible values for balances by using bins of low, medium, high.

**Example: Netflix Ratings Data (2006 competition)**

Approx. 100 million ratings from 480k users, with 17.5k movies.
Narrow data set because each row contains just four fields: user_id, movie_id, date, rating

We can widen narrow data by creating derived variables from rich columns, such as:

- ratings in first month
- ratings per month (after first month)
- total number of ratings
- proportions of {love, hate} ratings

## 10.2. Dimensionality Reduction

**Problems with High-Dimensional Data**

- Risk of correlation between input variables: many input variables are likely to correlate which mean scertain characteristics can be over-weighted (for clustering/memory-based reasoning; effect is more subtle for decision trees).
- Risk of overfitting: with too many input variables, model parameters are tailored closely to the training data and resulting models do not generalise well; NN regression is representative example
- Sparse data problem: imagine data as points in $n$-dimensional space, with one dimension for each input variable; sparse data results in isolated points without many neighbours; visualising relationships may highlight unpopulated area
  When using data sets with many variables, sparse data may deteriorate ability of data mining techniques to find interesting patterns.

### 10.2.1. Variable Selection

When dealing with high-dimensional problems, not all variables are necessary for accurate predictions. Variable selection therefore depends on relevance of variables:

- **Independence**: an input variable $x_j$ and output variable $y$ are independent if $\Pr(y|x_j) = \Pr(y)$, that is, $x_j$ is not relevant for determining $y$. We are interesting in quantifying a degree of dependence, for numeric we can estimate **linear correlation coefficients**, for nominal variables we compute **average mutual information**.
  - **Correlation**: statistical measure between variables; ranges between −1 to 1

$$\rho(x_j, y) = \frac{\sum_{i=1}^{m}(x_{i,j} - \overline{x}_j)(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_{i,j} - \overline{x}_i)^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

  - **Average Mutual Information**: given $x_j, y$ categorical data, give an estimate of dependence between input variable $x_j$ and target variable $y$; amount of information obtained for $y$ if we observed $x_j$

$$I(y; x_j) = H(y) - H(y|x_j)$$

$$\text{conditional entropy: } H(y|x_j) = -\sum_{y \in \text{Dom}(y)} \sum_{x \in \text{Dom}(x_j)} \Pr(x, y) \log(\Pr(x|y))$$

The interaction of each individual variable $x_j$ with $y$ is not necessarily informative about the way the variables interact with $y$. Generally, the set of $k$ best variables, e.g. most correlated, is not the same with the best set of variables of size $k$.

For many prediction problems, there is no optimal algorithm for variable selection.
There are many good heuristic methods detailed below.

**Exhaustive Feature Selection**

Given a set of variables, exhaustively try all combinations. Definitely creating the best model, but quickly becomes impractical for large amounts of input variables due to massive number of possible variable combinations.

**Selection using a target variable**

Technique of selecting variables for specific models, e.g. regression and decision trees, use a target variable $y$ to select the best subset $X$ of input variables.
Usually effective with a training/validation test set.

For very high-dimensional data sets, no. of instances might be relatively small vs. instances.
In this case, number of instances might be insufficient to create a validation test set.

**Sequential Feature Selection**
One variable is considered at a time for inclusion in the model.
The process of selecting a model can be separate from process of building the model:

- **Forward Selection**
  - start without any variables in the model
  - build a family models with one input variable per model
  - pick the input variable that fits best
  - repeat by adding one variable at a time until we reach a predefined max. number of variables or if adding a new variable does not improve the model
- **Backward Selection**: start with all variables initially included, remove one variable at a time

> **Note**
>
> - Separation of data set into training, validation, and test set is useful.
> - Once a family of models have been created, the best model is determined based on performance on the validation set.
> - Using forward selection with logistic regression to select variables and then use these variables as inputs to a neural network has resulted in strong predictive models.
> - For high-dimensional data sets, backward selection is impractical.

### 10.2.2. Variable Transformation

An **orthogonal matrix** $A$ is where $P^{\{-1\}} = P^T$.

#### 10.2.2.1. Principal Component Analysis

**Principal Component Analysis (PCA)**

Given an $m \times n$ input data matrix $\boldsymbol{X}$, PCA transforms $\boldsymbol{X}$ to an $m \times k$ output matrix $\boldsymbol{Y}$, that best represents $\boldsymbol{X}$ via the mapping $\boldsymbol{Y} = \boldsymbol{XP}$, where $k \leq n$. $\boldsymbol{P}$ is an $n \times k$ orthogonal projection matrix.

- Using $k$ significantly smaller than $n$, i.e. when $\boldsymbol{Y}$ has a smaller no. of dimensions than $\boldsymbol{X}$, PCA performs **dimensionality reduction**, i.e. data compression.
- Original data points are represented using a different set of variables, i.e. PCA performs **variable transformation**.
- The matrix $\boldsymbol{P}$ is computed so that the reconstruction error $\sum_{i=1}^{n} \|x_i - \boldsymbol{P}\boldsymbol{P}^T x_i\|^2$ is minimised.

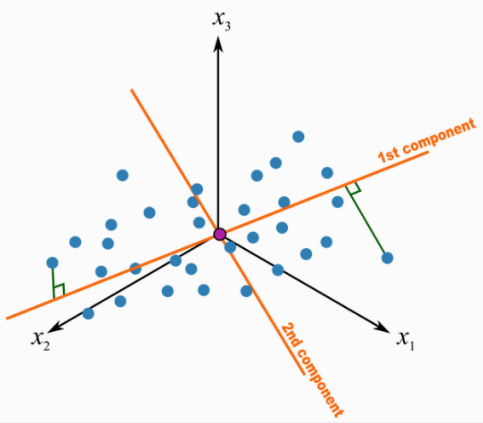



Figure 94: Principal Components

Each column of $\boldsymbol{P}$ is a **principal component**. Since the projection matrix $\boldsymbol{P}$ is orthogonal ($\boldsymbol{P}^{\{-1\}} = \boldsymbol{P}^T$), the principal components are perpendicular, i.e. have orthogonal directions.

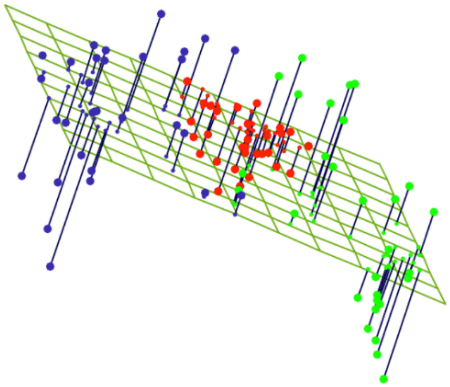


Figure 95: Projection to Lower Dimensional Subspace

The principal components represent the directions of the reduced subspace and are linear combinations of the original variables.

**First principal component**: variant of the best-fit linear regression line where, instead of vertical distances to fit the best line, orthogonal distances to the best-fit line are used.

- The residuals between the data points and the first principal component are vectors with a distance and direction orthogonal to the principal component line.
- Hence, these residuals can be plotted as points in the space.
- The principal component of the residuals is the second principal component. Following the same idea, we may compute the remaining principal components.

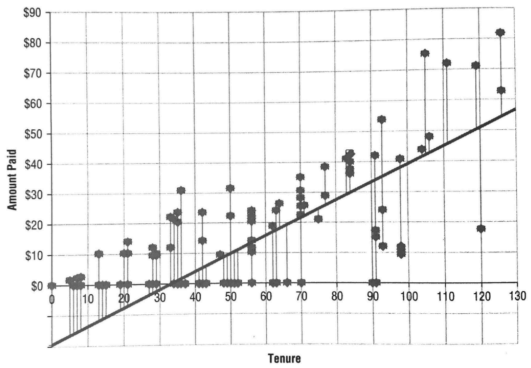



Figure 96: Best-fit linear regression minimising sum of squared vertical dist.

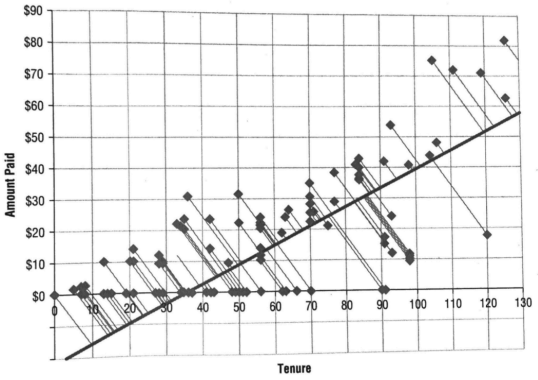



Figure 97: First principal component is the line minimising sum of squared orthogonal distances from each data point to the line

PCA finds directions (principal components) along which the data is spread the most from the mean, i.e. capturing the greatest variability. The greatest variance in the data lies on the first principal component, second greatest on the 2nd, etc.

**PCA can be implemented by** computing the eigenvectors (principal components) of the covariance matrix because they specify the orthogonal directions capturing most variability in the training data.

- The first eigenvector accounts for lagest possible variance and is associated with the largest eigenvalue. Second is for second largest, etc.
- Hence, PCA computes eigenvectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ corresponding to the largest eigenvalues $\lambda_1, \ldots, \lambda_k$ of the covariance matrix $\boldsymbol{\Sigma}$.

> **Note**
>
> - PCA can be considered as a rotation of the axes of the original coordinate system to new orthogonal axes (principal components) which coincide with directions of the maximum variation of original observations, i.e. PCA computes projections with maximum variance.
> - Once projection matrix $\boldsymbol{P}$ has been computed, each original data point can be projected in the subspace specified by the principal components.
> - PCA reduces the number of variables.
> - Since principal components are orthogonal to each other, the model is easy to interpret.
> - Variables are independent due to orthogonality.
> - Usually considered an unsupervised learning technique.
> - Presents disadvantage that in the resulting subspace, there are aspects of the original space which are not captured.

#### 10.2.2.2. Singular Value Decomposition

A non-negative number $\boldsymbol{\sigma}$ is a **singular value** of an $m \times n$ matrix $\boldsymbol{A}$ if there exist unit-length vectors $\boldsymbol{u}$ (**left-singular vector**) and $\boldsymbol{v}$ (**right-singular vector**) such that:

$$\boldsymbol{A}\boldsymbol{v} = \boldsymbol{\sigma}\boldsymbol{u} \quad \text{and} \quad \boldsymbol{A}^T\boldsymbol{u} = \boldsymbol{\sigma}\boldsymbol{v}$$

**Singular Value Decomposition** is a factorisation of an $n \times m$ matrix $\boldsymbol{A}$, given by:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

- Generalisation of eigendecomposition of a square matrix to any $n \times m$ matrix.
- $\boldsymbol{U}$: $m \times m$ orthogonal matrix whose columns are called left singular vectors.
- $\boldsymbol{\Sigma}$: $m \times n$ diagonal matrix whose diagonal entries $\sigma_{i,i}$ are the singular values.
- $\boldsymbol{V}$: $n \times n$ orthogonal matrix whose columns are called right singular vectors.

> **Note**
>
> In the context of text mining:
> - $\boldsymbol{A}$: document-term matrix describing number of occurrences of terms in documents.
> - Rows represent documents while columns correspond to terms.
> - The $k$ largest singular values and the corresponding singular vectors from $\boldsymbol{U}$ and $\boldsymbol{V}$ give an approximation of $\boldsymbol{A}$.
> - Each row $\boldsymbol{u}_i$ of $\boldsymbol{U}$ corresponds to a document and each column $\boldsymbol{v}_j$ of $\boldsymbol{V}$ to a term.

**Latent Semantic Indexing** is a natural language processing technique for analysing relationships between documents and terms by accounting for semantic structure in documents.
- Uses SVD to reduce the number of rows while preserving the similarity structure among columns.
- Reduces the $m \times n$ matrix to an $m \times k$ matrix, where $k < n$.
- Terms with similar meaning are expected to be merged in the same dimension (e.g. principal component) after reducing dimensionality

#### 10.2.2.3. Projection Pursuit Regression

**Projection Pursuit Regression** is a non-linear transformation of linear combinations of variables by finding the most interesting projections of the data:

$$\hat{y} = \sum_{j=1}^{k} w_j h_j(\boldsymbol{\alpha}_j^T \boldsymbol{x})$$

Where $k$ is the number of new variables (typically $\ll n$: original no. of variables), $\boldsymbol{\alpha}_j^T \boldsymbol{x}$ is the projection of vector $\boldsymbol{x}$ to the $j$-th weight vector $\boldsymbol{\alpha}_j$, $h_j$ is a non-linear function of this scalar projection, and $w_j$ are scalar weights.

Both vectors $\boldsymbol{\alpha}_j^T$ and $\boldsymbol{x}$ are $n$-dimensional and result in a scalar inner product.

> **Note**
>
> - Projection pursuit methods can be proven to have the same ability of estimating an arbitrary function with neural networks, but are not widely used.
> - The functions $h_j$ are not constrained to have a specific form as in neural networks, but are usually selected based on the idea of smoothing.
> - In the case of neural networks, the functions are chosen to be of the form:
>
> $$h_j(t) = \frac{1}{1 + e^{-t}}$$
>
> - Procedures for determining model components, i.e. weights $w_j$, projection function $h_j$, and projection directions $\boldsymbol{\alpha}_j$ are complex and algorithm-dependent, but high-level idea is gener.
> - Fitting process is complex from computational viewpoint so projection pursuit methods are not practical for massive high-dimensional data sets.
> - PCA is a special case of projection pursuit regression where projection follows the direction with the variance.

# 11. Appendix

## 11.1. Glossary

## 11.2. Figures

TODO

Table 1: Contact Lens Data Set [WFH3, Table 1.1]

TODO

Table 2: Nominal Weather Data Set [WFH3, Table 1.2]

TODO

Table 3: Numeric Weather Data Set [WFH3, Table 1.3]

TODO

Table 4: CPU Performance Data Set [WFH3, Table 1.5]

TODO

Figure 98: Family Tree [WFH3, Figure 2.1]

TODO

Figure 99: Attribute-Relation File Format [WFH3, Figure 2.2]