MA0218 Introduction To Data Science And Artificial Intelligence Notes

Hankertrix

April 29, 2025

Contents

1 Definitions

1.1 Minkowski distance

The Minkowski distance of order $p, p \in \mathbb{Z}$ between two points X and Y is given as:

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Where:

$$X = (x_1, x_2, \ldots, x_n) \in \mathbb{R}$$
$$Y = (y_1, y_2, \ldots, y_n) \in \mathbb{R}$$

1.2 Mean (central tendency) ($\bar{x}$)

The mean is the average of the data.

It is defined as:

$$\text{Mean} = \frac{\text{Sum of data}}{\text{Count of data}}$$
$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

Where:

- $\bar{x}$ is the mean of the data set
- x_1 is the first item in the data set
- x_2 is the second item in the data set
- x_n is the n^{th} item in the data set
- n is the number of items in the data set

1.3 Standard deviation (dispersion) (σ)

The standard deviation is the average deviation from the mean.

It is defined as:

$$\text{Standard deviation} = \frac{\text{Sum of deviation}}{\text{Count of data}}$$

$$\sigma = \sqrt{\frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}{n}}$$

Where:

- σ is the standard deviation of the data set
- $\bar{x}$ is the mean of the data set
- x_1 is the first item in the data set
- x_2 is the second item in the data set
- x_n is the n^{th} item in the data set
- n is the number of items in the data set

1.4 Median (central tendency)

The median is the middle value of the data.

It is defined as the marker to divide the data in half, or mathematically:

$$P\left(x \leq x_M\right) = P\left(x \geq x_M\right) = 0.5$$

Where:

- P is the probability
- x is the value of an item
- x_M is the median value of the data

1.5 Quantiles (dispersion)

A quantile tells you the distribution of the data.

It is defined as the markers to divide the data in the form, 25:50:25, i.e.

$$P\left(x \leq x_{Q1}\right) = 0.25, P\left(x \geq x_{Q2}\right) = 0.25$$

$$P(x_{Q1} \leq x \leq x_{Q2}) = 0.5$$

Where:

- P is the probability
- x is the value of an item
- x_{Q1} is the value of the item at the first quartile (25th percentile)
- x_{Q2} is the value of the item at the second quartile (75th percentile)

1.6 Correlation coefficient (ρ)

The correlation coefficient tells you the dependence of one variable on the other variable.

It is defined as:

$$\frac{\text{Co-Variance}}{\text{Standard Deviation Product}}$$

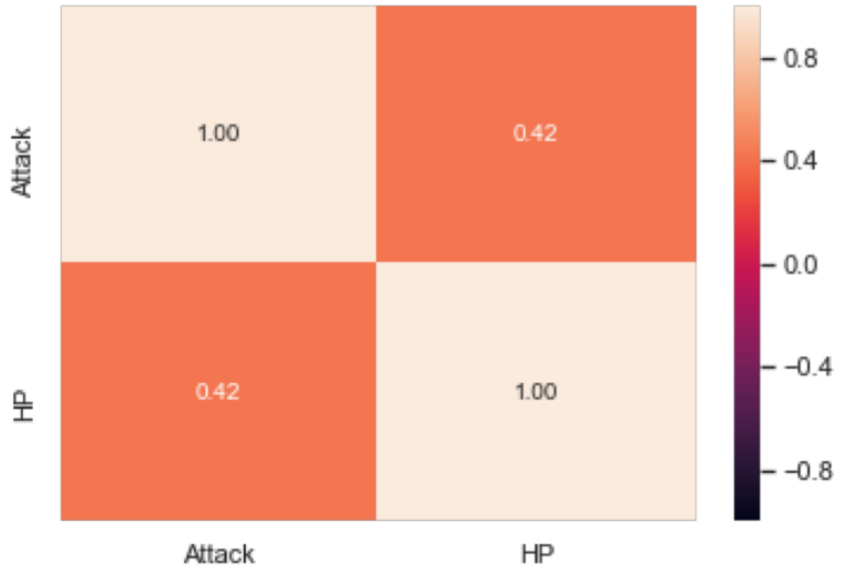
$$\rho_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}}$$

Where:

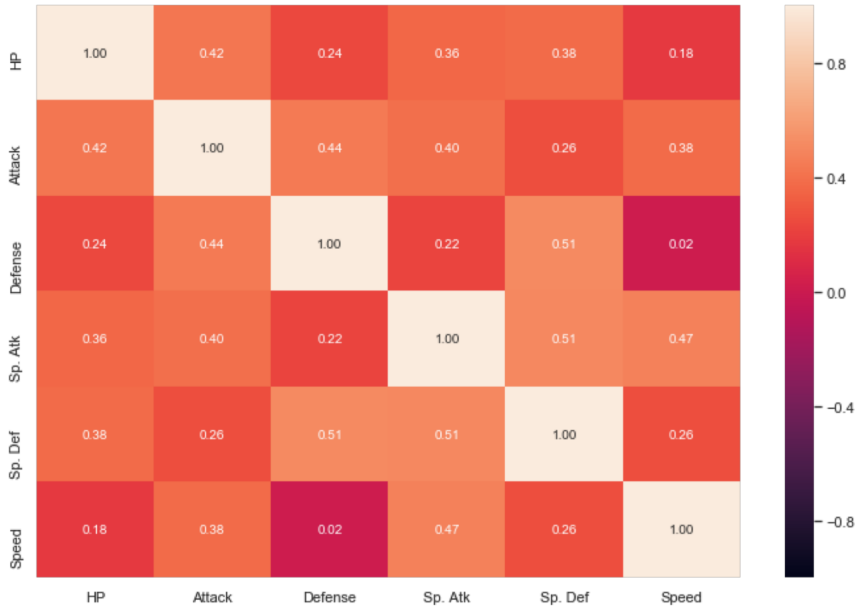
- ρ_{xy} is the correlation coefficient between variables x and y
- x_i is the value of an item in the x data set
- $\bar{x}$ is the mean of the data set for x
- y_i is the value of an item in the y data set
- $\bar{y}$ is the mean of the data set for y

Correlation coefficient	Dependence
$\rho = 0$	No dependence
$\rho = 1$	Perfect positive correlation
$\rho = -1$	Perfect negative correlation

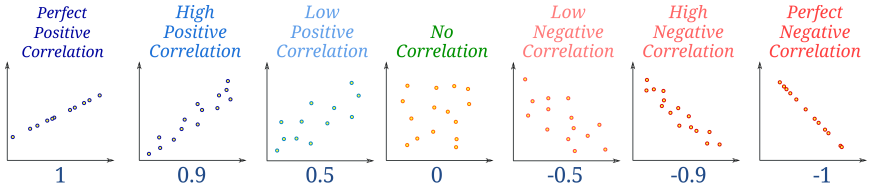
1.6.1 2 variable correlation plot



1.6.2 Mutual correlation plot



1.6.3 Examples



1.7 Residual sum of squares (RSS)

The residual sum of squares is defined as:

$$RSS = \sum (y_{actual} - y_{predicted})^2$$

Where:

- RSS is the residual sum of squares
- y_{actual} is the actual value from the data set
- $y_{predicted}$ is the value predicted by the model

1.8 Mean-squared error (MSE)

The lower the mean-squared error (MSE), the better the model. Mean-squared error is defined as:

$$MSE = \frac{RSS}{n}$$

Where:

- MSE is the mean-squared error
- RSS is the residual sum of squares
- n is the number of items in the data set

1.9 Total sum of squares (TSS)

The total sum of squares is sum of the difference between a data point and the mean of the data set, squared. It is defined as:

$$\begin{aligned} TSS &= \sum_{i=1}^{n} (x_i - \bar{x})^2 \\ &= (x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2 \end{aligned}$$

Where:

- TSS is the total sum of squares
- n is the total number of items in the data set
- x is a data point
- $\bar{x}$ is the mean of the data set

1.10 Variance (VAR)

The variance is the expected deviation from the mean, squared. It measures how far a set of numbers is spread from their average value. It is defined as:

$$VAR = \frac{TSS}{n}$$

Where:

- VAR is the variance
- TSS is the total sum of squares
- n is the number of items

1.11 Explained variance (R^2)

The higher the explained variance (R^2), the better the model. Explained variance is defined as:

$$R^2 = 1 - \frac{MSE}{VAR}$$

Where:

- R^2 is the explained variance
- MSE is the mean-squared error
- VAR is the variance

1.12 Gini Index (Gini Impurity)

The Gini Index tells you the chance of misclassification when you are in a specific partition of your data. It is defined as follows:

$$\begin{aligned} Gini &= \sum_{i=1}^{n} \frac{x_i}{N}\left(1 - \frac{x_i}{N}\right) \\ &= \frac{x_1}{N}\left(1 - \frac{x_1}{N}\right) + \frac{x_2}{N}\left(1 - \frac{x_2}{N}\right) + \cdots + \frac{x_n}{N}\left(1 - \frac{x_n}{N}\right) \end{aligned}$$

Where:

- $Gini$ is the Gini Index
- x is the number of items belonging to a class in the data partition
- N is the total number of items in your data set

A simpler equation for two items is shown below:

$$Gini = \frac{x}{N}\left(1 - \frac{x}{N}\right) + \frac{y}{N}\left(1 - \frac{y}{N}\right)$$

Where:

- $Gini$ is the Gini Index
- x is the number of items belonging to one class in that node or partition
- N is the total number of items in the that node or partition
- y is the number of items belonging to the other class in that node or partition

1.13 Within-cluster sum of squares

The within-cluster sum of squares is calculated as follows:

- Take the point that represents the middle of the cluster as a reference point.
- For each point **in** the cluster, calculate the distance between the point and the reference point (the point in the middle of the cluster) and square it.
- Sum up the distances for all the points, and you have the within-cluster sum of squares.

1.14 Classification accuracy

The classification accuracy is the number of items classified correctly over the total number of items. It is defined as:

$$\text{Classification accuracy} = \frac{T_p + T_n}{n}$$

Where:

- T_p is the number of true positives
- T_n is the number of true negatives
- n is the total number of items

1.15 Classification accuracy metrics

The easy way to remember these metrics is to remember that the equations are always divided by themselves plus the inverse of themselves.

1.15.1 True positive rate (tpr)

$$tpr = \frac{T_p}{T_p + F_n}$$

Where:

- tpr is the true positive rate
- T_p is the number of true positives
- F_n is the number of false negatives

1.15.2 True negative rate (tnr)

$$tnr = \frac{T_n}{T_n + F_p}$$

Where:

- tnr is the true negative rate
- T_n is the number of true negatives
- F_p is the number of false positives

1.15.3 False positive rate (fpr)

$$fpr = \frac{F_p}{F_p + T_n}$$

Where:

- fpr is the false positive rate
- F_p is the number of false positives
- T_n is the number of true negatives

1.15.4 False negative rate (fnr)

$$fnr = \frac{F_n}{F_n + T_p}$$

Where:

- fnr is the false positive rate
- F_n is the number of false negatives
- T_p is the number of true positives

1.16 Precision

Precision is defined as:

$$P = \frac{T_p}{T_p + F_p}$$

Where:

- P is the precision
- T_p is the number of true positives
- F_p is the number of false positives

1.17 Recall

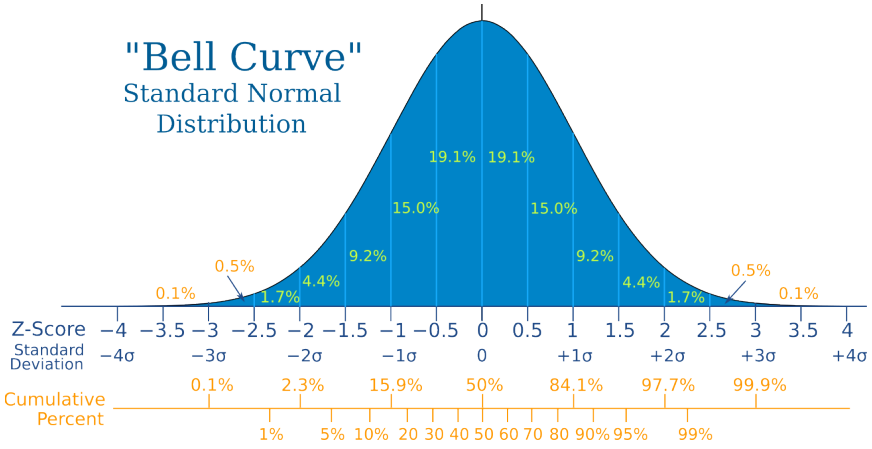
Recall is defined as:

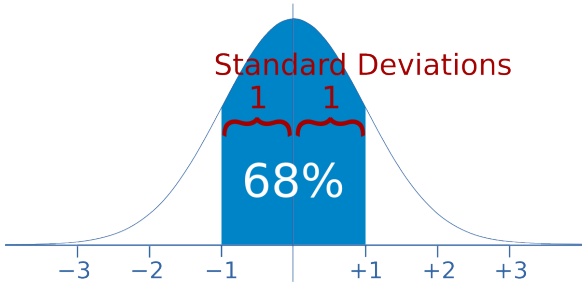
$$R = \frac{T_p}{T_p + F_n}$$

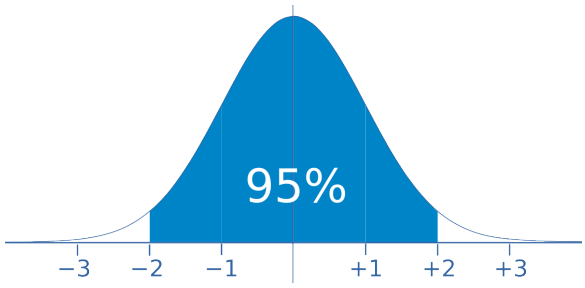
Where:

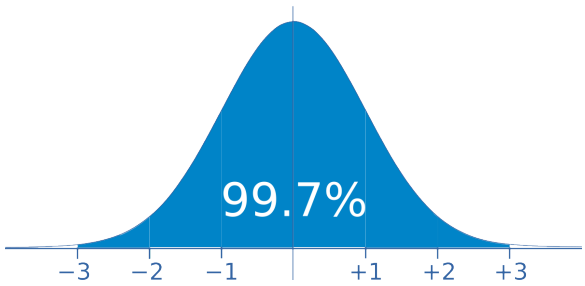
- R is the recall
- T_p is the number of true positives
- F_p is the number of false positives

1.18 Normal distribution







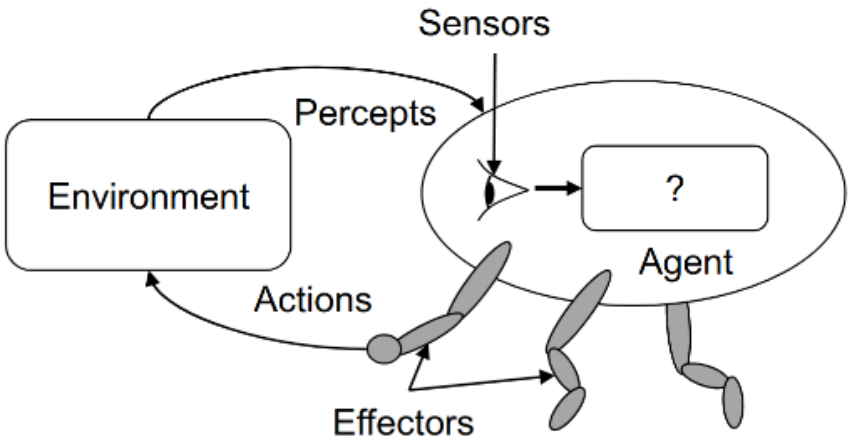


1.19 Artificial Intelligence (AI)

AI is intelligence demonstrated by machines, in contrast to the *natural intelligence (NI) displayed by humans and other animals.

1.20 Agent

An **agent** is an entity that **perceives** through sensors, like eyes, ears, cameras, and infrared range sensors, and **acts** through effectors, such as hands, legs and motors.



1.21 Rational action

A rational action is an action that maximises the expected value of an objective **performance** measure given the perception sequence to date.

2 Data science pipeline

2.1 Practical motivation

How to identify a data science case?

- What is the real-life problem?
- Can you relate the problem to data?
- Would data help you in practice?

2.1.1 Sample collection

How to effectively sample real data?

- How to collect the relevant data?
- Does the data match the problem?
- Does the data represent reality?

2.2 Problem formulation

How to intelligently construct a problem?

- What is the data science problem?
- How do you formulate it using data?
- How do you solve it using the data?

2.2.1 Data preparation

How to prepare raw data for analysis?

- How to prepare the relevant data?
- Is the data clean enough to analyse?
- Is the data structured for analysis?

2.3 Statistical description

How to succinctly represent the data?

- How to clearly describe the data?
- How do you summarise the data?
- Which vital statistics are relevant?

2.3.1 Exploratory analysis

How to gain basic insight from data?

- How to explore the acquired data?
- How to effectively mine the data?
- How to compute the vital statistics?

How to intelligently explore acquired data?

- What are the variables in the data?
- How to characterise the variables?
- How to find relations between them?

2.4 Pattern recognition

How to find intrinsic insight from the data?

- How to identify structure in data?
- Can you see the known patterns?
- Can you discover unknown traits?

2.4.1 Analytic visualisation

How to represent the data for humans?

- How to clearly visualise the data?
- How to visually represent statistics?
- How to highlight "interesting" traits?

2.5 Machine learning

How to efficiently learn from the data?

- How to learn from the data?
- Can you formulate the "learning"?
- Can you automate the "learning"?

2.5.1 Algorithmic optimisation

How to optimally learn from the data?

- How to form "learning" algorithms?
- How to reduce errors in learning?
- How to generalise the algorithms?

2.6 Statistical inference

How to confidently infer from the data?

- How to draw conclusion from data?
- Can you generalise the "learning"?
- Can you estimate the confidence?

2.6.1 Information presentation

How to communicate your data analysis?

- How to present analysis outcomes?
- How to present descriptive analysis?
- How to present inferential analysis?

2.7 Intelligent decision

How to solve a real-life problem by data?

- How to make decisions in practice?
- Can you decide based on the data?
- Can you optimise the outcomes?

2.7.1 Ethical consideration

How to responsibly work in data science?

- How to conform to ethical values?
- Does the analysis violate legality?
- Does the decision violate ethics?

3 Data science problems and solutions

3.1 How much? How many?

The model should **predict** a **numeric** value.
Examples:

- What is the expected sales of the next game of this game franchise?
- Is it profitable to make the sequel?

3.1.1 Solution: Regression

Try to find the relationship of sales of the games with other variables, like graphics quality, genre, etc.

$$\text{Model: Sales} = f(\text{Variables})$$

Examples:

- Linear regression models
- Tree models for regression
- Neural network for regression

3.2 Is it type A or type B?

The model should **predict** a **category** or a **class** for an object.
Examples:

- What is the chance that a student will get into NTU in AY2019-2020?
- Will an application be successful?

3.2.1 Solution: Classification

Try to find the probability of getting admitted to NTU in terms of other variables, like scores, gender, etc.

$$\text{Model: } \mathcal{P}(\text{Admit}) = f(\text{Variables})$$

Examples:

- Logistic Regression Model
- Tree models for classification
- Neural network for classification

3.3 How is this organised?

The model should **detect** the **structure** of an object.
Examples:

- Is there any structure apparent within the FairPrice customers?
- Which customer group to target?

3.3.1 Solution: Clustering

Try to find groups of data points that are close together but far from the other groups of points. How close and how far depends on the "distance", like the Minkowski distance, as shown below:

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Some examples of "distances":

- Euclidean distance
- Jaccard distance

Examples of clustering models:

- k-Means algorithm for clustering
- Hierarchical model for clustering

3.4 Is it weird behaviour?

The model should **detect anomalies** in the data.
Examples:

- Is this Boeing engine behaving unusually during the flight?
- Is the engine still safe to operate?

3.4.1 Solution: Anomaly detection

Try to find deviations in the data compared to the regular pattern observed through the data model. The deviations depend on the model.
Examples:

- Cluster-analysis based detection
- Nearest neighbour detection model
- Support vector-based detection

3.5 What should be done next?

The model should **detect** the next **action** to be done.
Examples:

- Should the car brake at the yellow light, or should the car accelerate instead?
- Which action will be rewarded?

3.5.1 Solution: Adaptive learning

Try to model a profit or loss function depending on the given state, and try to maximise the profit or minimise the loss, respectively.

$$\text{Optimisation function} : f(\text{State, Variables})$$

Examples of using the reinforcement learning approach:

- Monte-Carlo
- State-Action-Reward
- Q-Learning
- Deep Reinforcement

4 Common data types

4.1 Structured data

- Highly organised
- Easy to analyse

4.1.1 Numeric data

- Highly organised data
- Clearly defined variables
- Easy to mine and analyse
- Numeric continuous variables

Possible sources:

- Spreadsheets (Excel, CSV)
- Standard SQL databases
- Sensors and devices

4.1.2 Categorical data

- Highly organised data
- Clearly defined variables
- Easy to mine and analyse
- Factor, level or class variables

Possible sources:

- Spreadsheets (Excel, CSV)
- Standard SQL databases
- Sensors and devices

4.1.3 Mixed data

- Highly organised data
- Clearly defined variables
- Easy to mine and analyse
- Numeric and categorical

Possible sources:

- Spreadsheets (Excel, CSV)
- Standard SQL databases
- Sensors and devices

4.1.4 Time series data

- Highly organised data
- Clearly defined variables
- Easy to mine and analyse
- Numeric with timestamps

Possible sources:

- Stock and equity markets
- Weather data over time
- Prices and promotions

4.1.5 Network data

- Highly organised data
- Clearly defined variables
- Easy to mine and analyse
- Numeric with timestamps

Possible sources:

- Social networks and the web
- Transport networks (MRT)
- Financial transactions

4.2 Unstructured data

- Highly unorganised
- Contextual

4.2.1 Text data

- Highly unorganised data
- Non-obvious variables
- Highly context-sensitive
- Words, phrases and emoticons

Possible sources:

- Social networks and the web
- Text messages or WhatsApp
- Books, wikis and documents

4.2.2 Image data

- Highly unorganised data
- Non-obvious variables
- Highly context-sensitive
- Words, phrases and emoticons

Possible sources:

- Social networks and the web
- Mobile phone cameras
- Blogs, wikis and documents

4.2.3 Video data

- Highly unorganised data
- Non-obvious variables
- Highly context-sensitive
- Images, frames and objects

Possible sources:

- YouTube and social media
- Video messages and calls
- Mobile phone cameras

4.2.4 Voice data

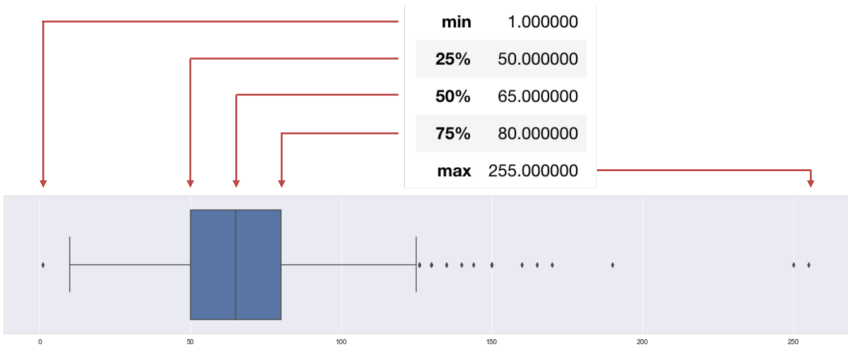
- Highly unorganised data
- Non-obvious variables
- Highly context-sensitive
- Voice signals and waves

Possible sources:

- Songs and social media
- Microphones and cameras
- Recordings and announcements

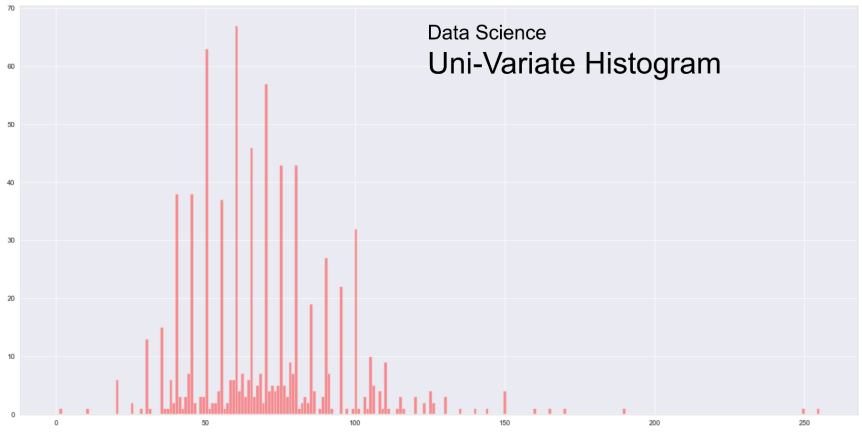
5 Plots

5.1 Uni-variate box-plot

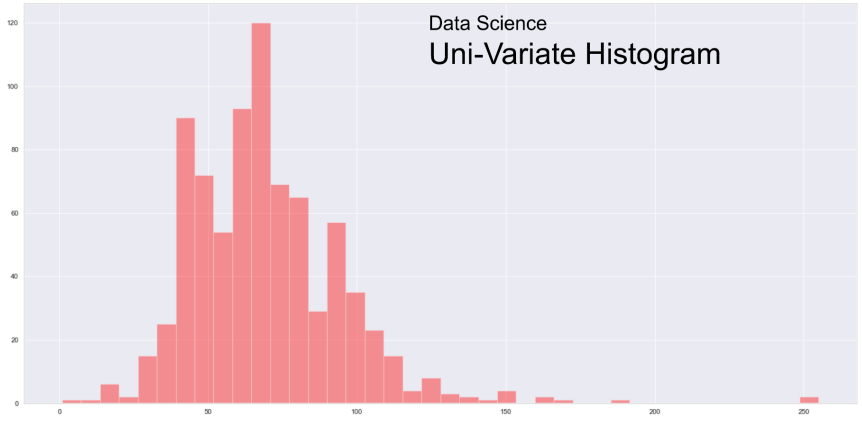


5.2 Uni-variate histogram

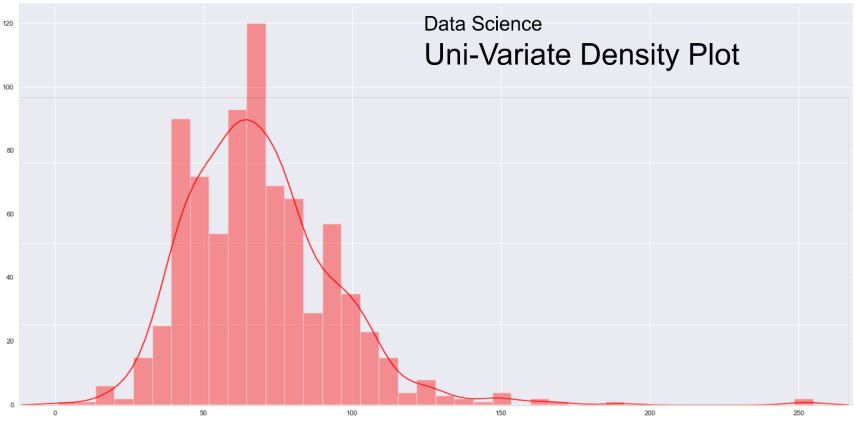
5.2.1 Variation 1



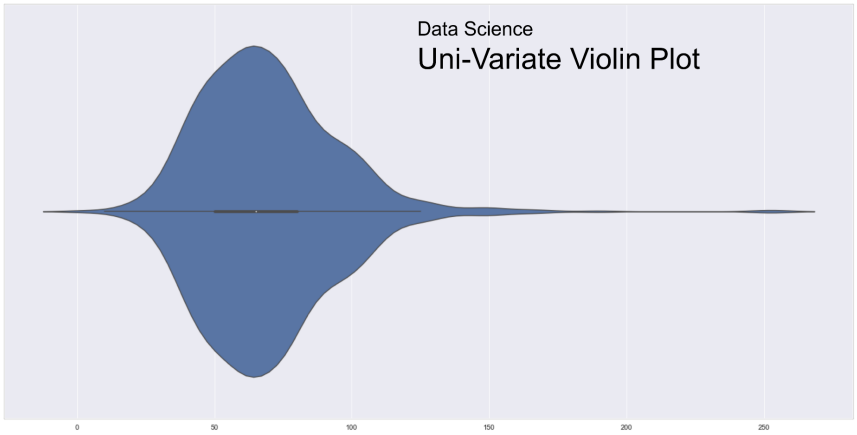
5.2.2 Variation 2



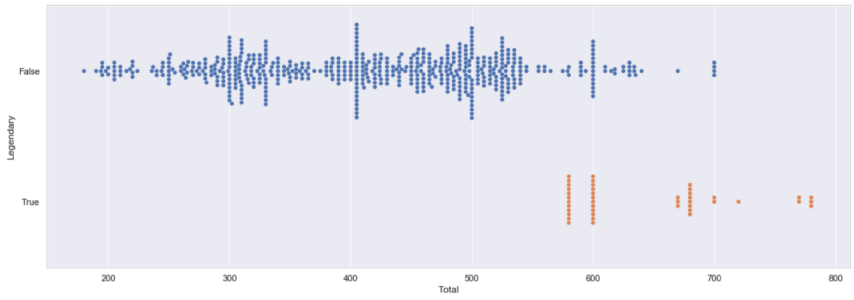
5.3 Uni-variate density plot



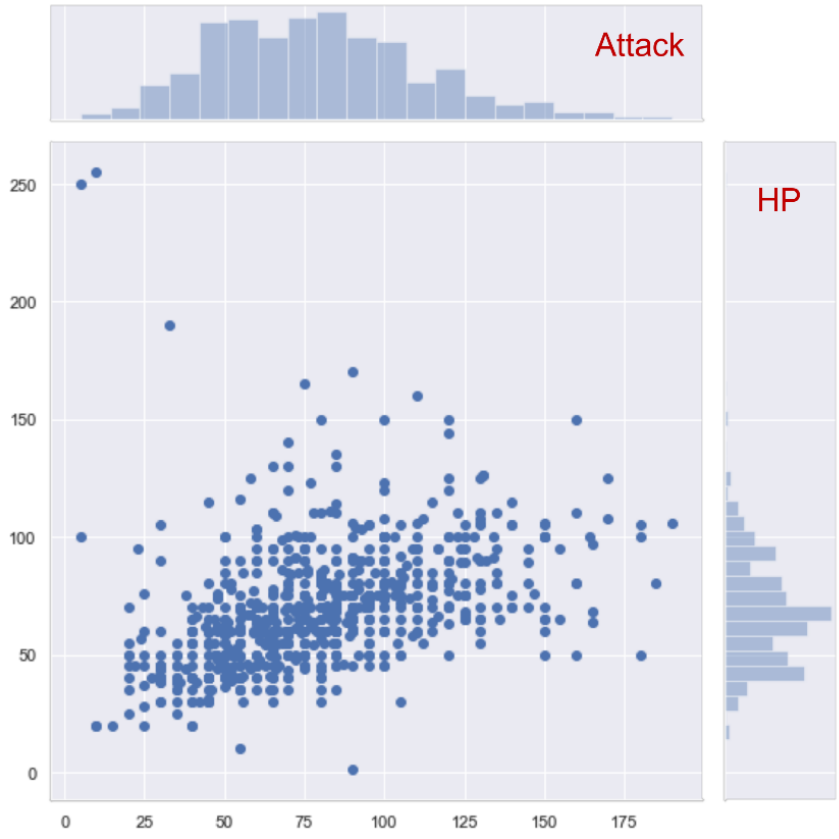
5.4 Uni-variate violin plot



5.5 Uni-variate swarm-plot



5.6 Bivariate joint plot



6 Machine learning

How to optimally learn from the data?

- What do we know about the problem?
- Does the data provide hints?
- Can we learn from the hints in the data?
- Can we test how well we have learnt?

6.1 Supervised learning

- Regression
- Classification

6.2 Unsupervised learning

- Clustering
- Anomaly detection

6.3 Numeric prediction problem

- How much?
- How many?

6.3.1 Regression

$$\text{Model: Total} = f(\text{Variables})$$

For example:

- Given some Pokémon to train.
- Learn the model for total points.
- Predict the estimated total for the others.

6.4 Classes prediction problem

Is it type A or type B?

6.4.1 Classification

$$\text{Model: } P(\text{Legend}) = f(\text{Variables})$$

For example:

- Given some Pokémon to train.
- Learn the model for legendary Pokémon.
- Determine the class for others.

6.5 Structure detection problem

How is this organised?

6.5.1 Clustering

Grouping depends on "distance".
For example:

- Given the distance on all Pokémon.
- Find the optimal groups in the data.
- Justify the interpretation of the groups.

6.6 Anomaly detection problem

Is it weird behaviour?

7 Uni-variate regression

- Are variables mutually dependent?
- How to find the relations between them?
- How to predict one using the other?

7.1 Split the data set into 2

1. A training data set to train the model.
2. A testing data set to test the model's performance.

The objectives:

- Learn the relationship from the training data set.
- Try to predict the total on the testing data set.

7.2 Statistical modelling

7.2.1 Hypothesise a linear model

$$y = ax + b$$

Where:

- y is the dependent variable, like the ranking of Pokémon, for example
- a is a constant that we need to find
- x is the independent variable, like the HP of a Pokémon, for example
- b is another constant that we need to find

7.2.2 Steps

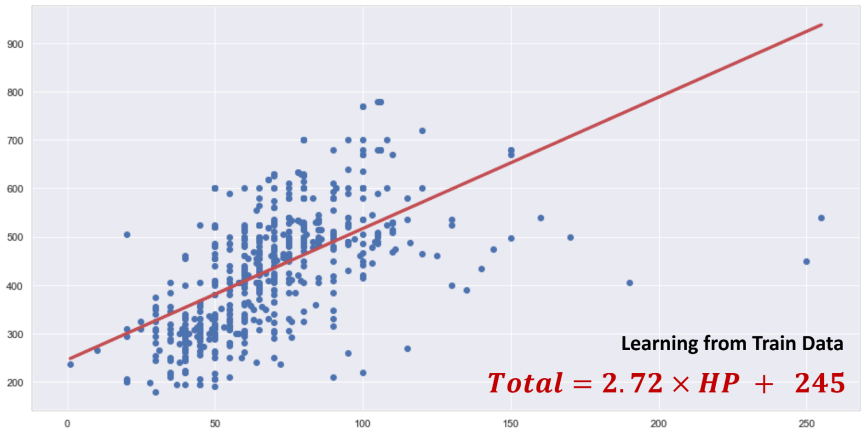
1. Guess parameters a and b in the model.
2. Predict the values of the **dependent variable**, y, in the training data.
3. Calculate the **errors** in the predicted value compared to the actual data.
4. Tune the parameters a and b to minimise errors.

7.2.3 Algorithmic optimisation

The cost function to minimise errors is the total-squared error:

$$J(a,b) = \sum (y - ax - b)^2$$

7.2.4 Learning from training data



7.2.5 Prediction on test data

7.3 Performance evaluation

The hypothesised model is a linear model:

$$y = ax + b$$

Where:

- y is the dependent variable, like the ranking of Pokémon, for example
- a is a constant that we need to find
- x is the independent variable, like the HP of a Pokémon, for example
- b is another constant that we need to find

7.3.1 Mean-squared error (MSE)

The lower the mean-squared error (MSE), the better the model. Mean-squared error is defined as:

$$MSE = \frac{1}{n}\sum(y - ax - b)^2$$

Where:

- MSE is the mean-squared error
- n is the number of items in the data set
- y is the dependent variable, like the ranking of Pokémon, for example
- a is a constant that we need to find
- x is the independent variable, like the HP of a Pokémon, for example
- b is another constant that we need to find

7.3.2 Explained variance (R^2)

The higher the explained variance (R^2), the better the model. Explained variance is defined as:

$$R^2 = 1 - \frac{\sum(y - ax - b)^2}{\sum(y - \bar{y})^2}$$

Where:

- R^2 is the explained variance
- y is the dependent variable, like the ranking of Pokémon, for example
- a is a constant that we need to find
- x is the independent variable, like the HP of a Pokémon, for example
- b is another constant that we need to find
- $\bar{y}$ is the mean of the data set for y

8 Binary classification

How to optimally learn from the data?

- Are variables mutually dependent?
- How to find relations between them?
- How to predict one using another?

8.1 Decision tree

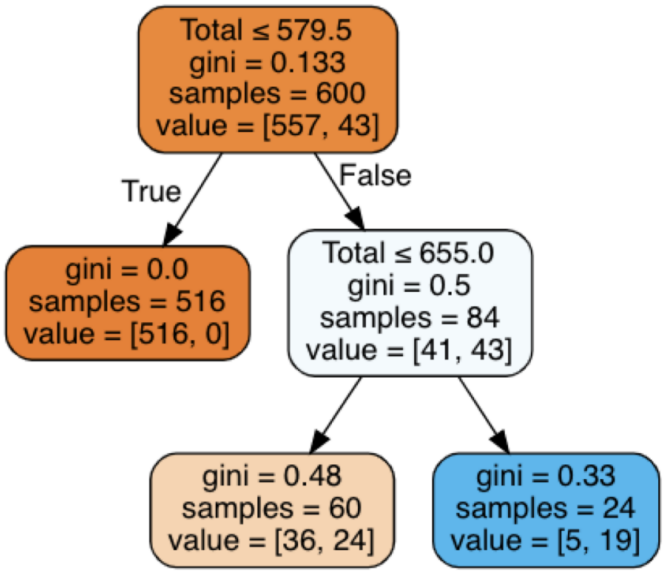
- A decision tree is the partitions made in the data space, which is methodically represented using consecutive binary decisions.
- The decision of the partition depends on the Gini Index, which tells you the chance of misclassification when you are in a specific node of a tree, or that specific partition of your data.

$$Gini = \frac{x}{n}\left(1 - \frac{x}{n}\right) + \frac{y}{n}\left(1 - \frac{y}{n}\right)$$

Where:

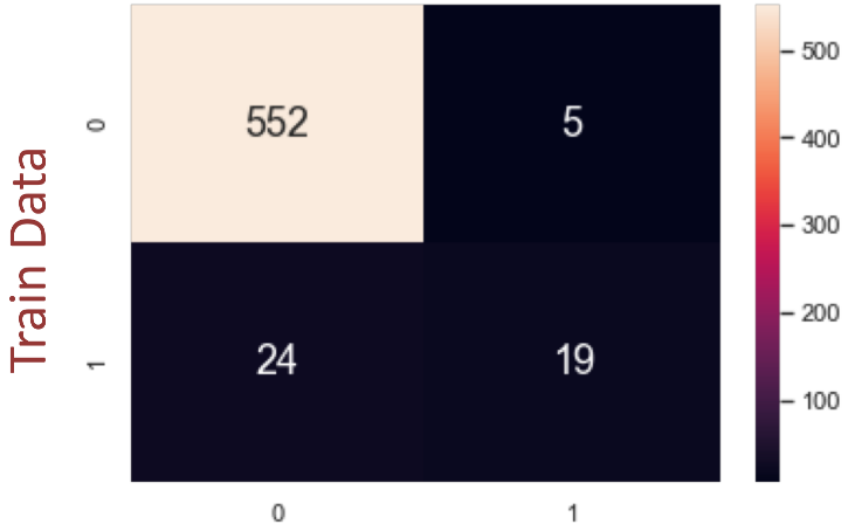
  - $Gini$ is the Gini Index
  - x is the number of items belonging to one class in that node or partition
  - n is the total number of items in the that node or partition
  - y is the number of items belonging to the other class in that node or partition

8.1.1 Example



8.1.2 Prediction using decision tree

The confusion matrix is shown below. The x-axis represents the predicted values, and the y-axis represents the actual values.



8.2 Classification accuracy

- The classification accuracy is the fraction of correct predictions.
  - A true positive (TP) is when true is predicted as true.
  - A true negative (TN) is when false is predicted as false.

It is given by:

$$\text{Classification accuracy} = \frac{\text{Number of true positives} + \text{Number of true negatives}}{\text{Total number of data points}}$$

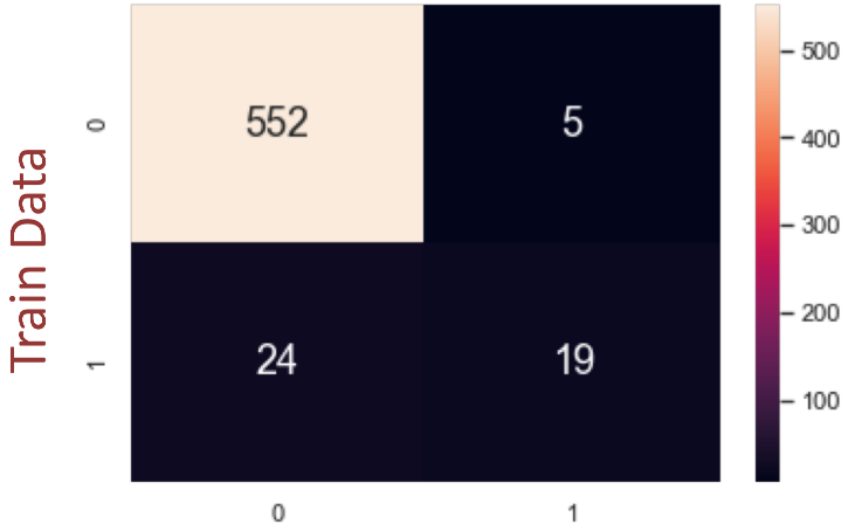
8.3 Classification errors

- A false negative (FN) is when true is predicted as false.
- A false positive (FP) is when false is predicted as true.

$$fpr = \frac{\text{Number of false positives}}{\text{Number of false positives} + \text{Number of true negatives}}$$

$$fnr = \frac{\text{Number of false negatives}}{\text{Number of false negatives} + \text{Number of true positives}}$$

8.3.1 Training data example



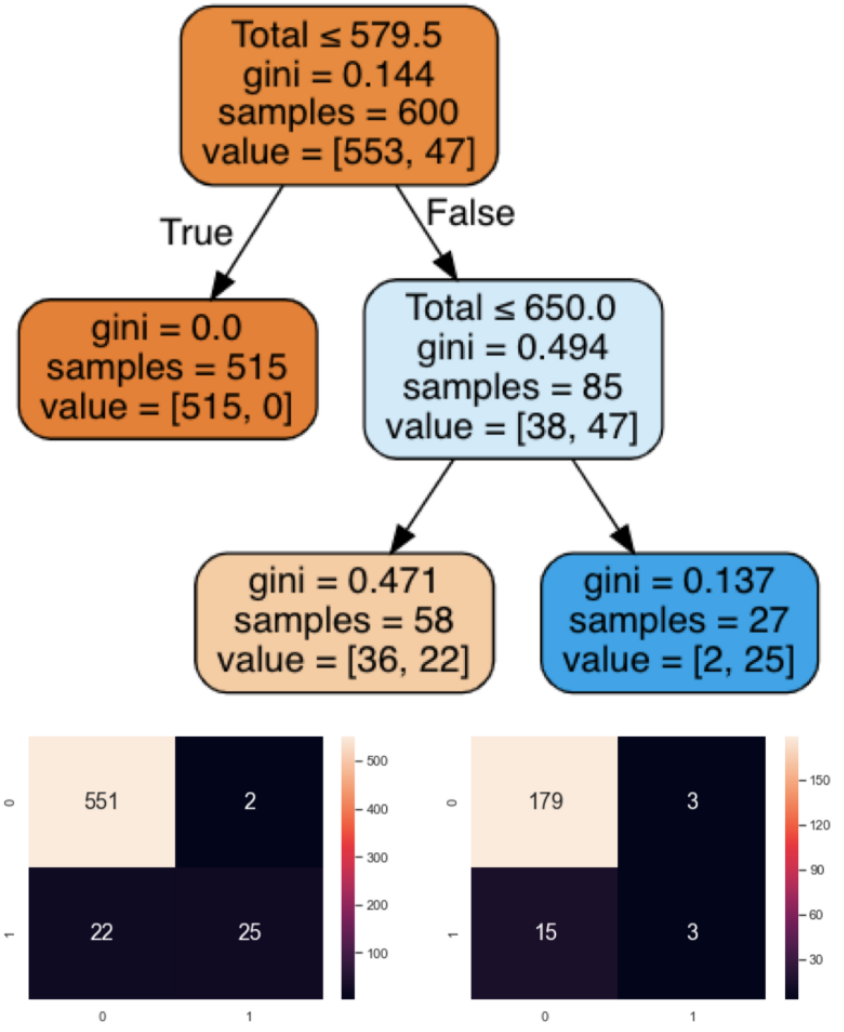
$$fpr = \frac{5}{557}, \quad fnr = \frac{24}{43}$$

8.3.2 Test data example

$$fpr = \frac{0}{178}, \quad fnr = \frac{13}{22}$$

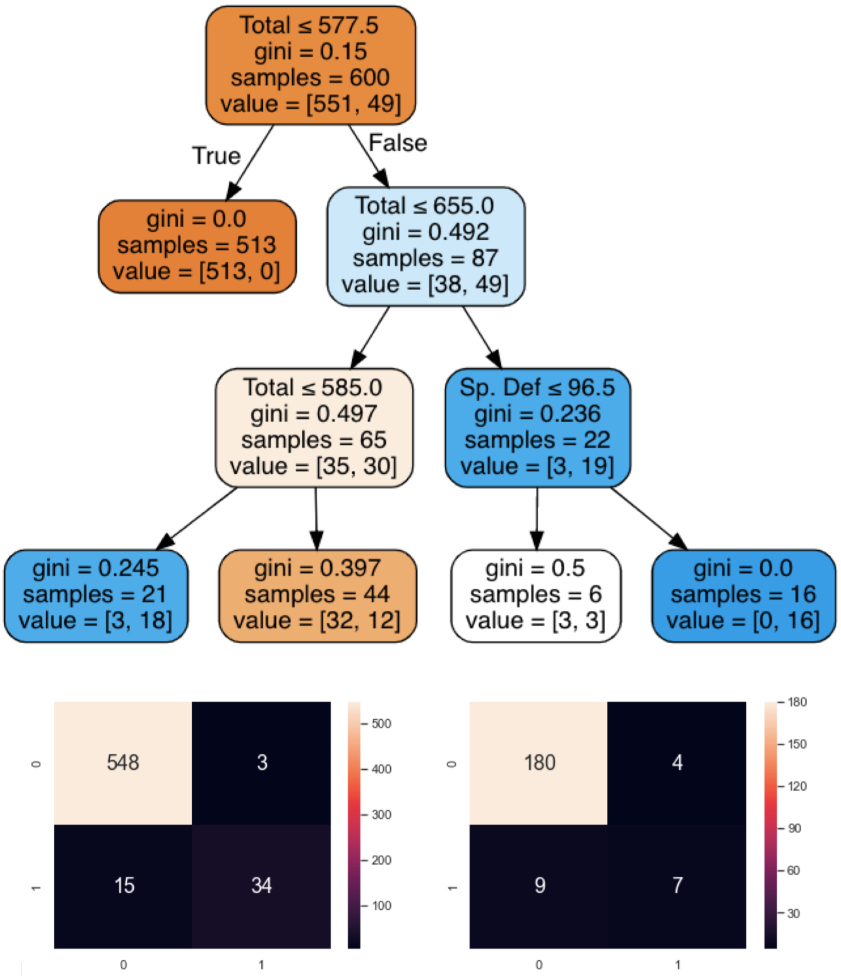
8.4 Two-level decision tree

The two-level decision tree and its corresponding confusion matrix is shown below.



8.5 Three-level decision tree

The three-level decision tree and its corresponding confusion matrix is shown below.



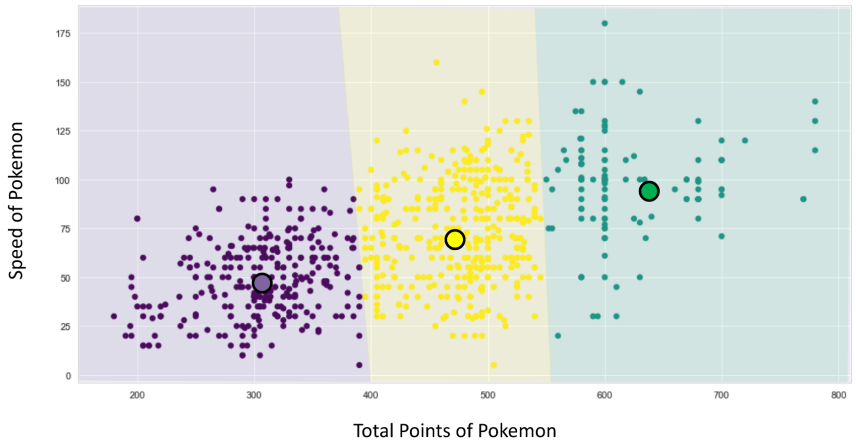
9 Pattern recognition

How to optimally learn from the data?

- Is there a pattern in the acquired data?
- How to learn the underlying pattern?
- How to exploit the pattern in the data?

9.1 K-means clustering

- **K** is the potential number of clusters.
- We will need to choose **K** cluster centroids from the data set, i.e. we will need to pick **K** number of points in the data set that is roughly in the middle of a cluster, to initialise the model.
- For each point in the data set, relabel the point according to the nearest centroid.
- For each cluster of data points, recompute the centroid of the cluster.



Machine learning questions:

- How many clusters are "visible"?
- Can we identify those clusters?
- What do the clusters signify?

Optimisation questions:

- What is the "optimal" cluster count?
- Can we justify the cluster count?
- What is a nice clustering metric?

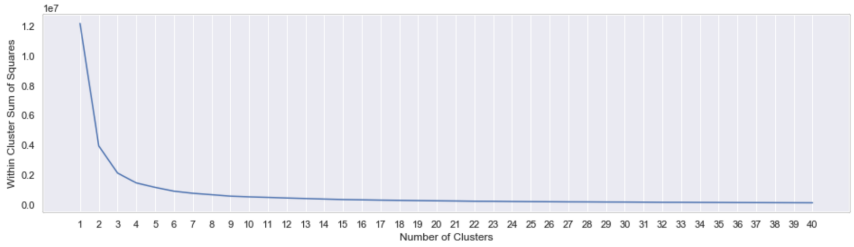
9.1.1 Example

Features	Total	Speed
Cluster 0	305.65	49.36
Cluster 1	622.57	97.08
Cluster 2	474.27	73.55

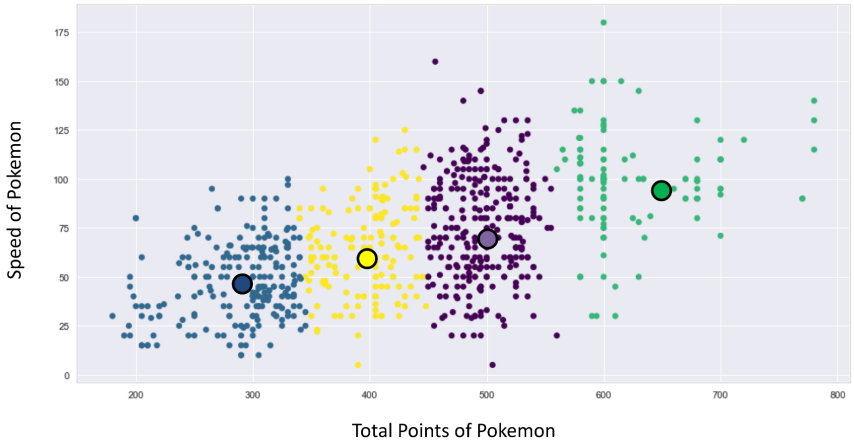
The within-cluster sum of squares is 2118651.

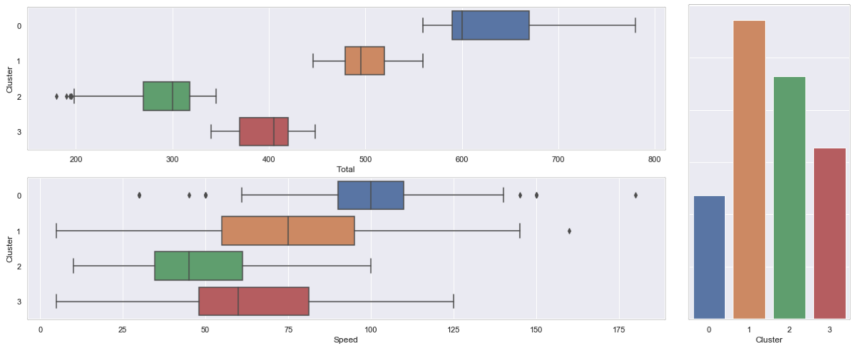
9.1.2 Determining the optimal number of clusters

For the above data set:



The guess for the optimal number of clusters is 4.

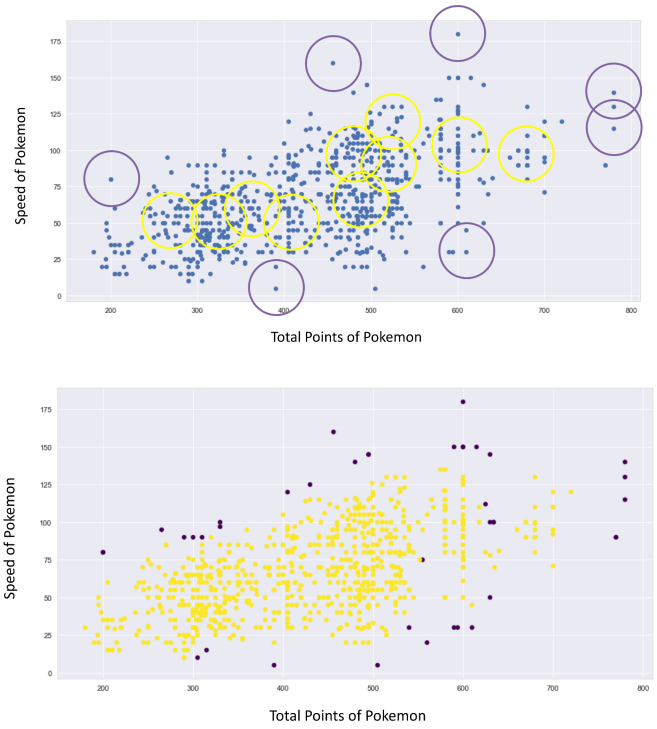




9.2 Local outlier factor (anomaly detection)

- **K** is the total number of neighbours.
- **d** is the fraction of anomalies in the data.
- For each point in the dataset, find **K** nearest neighbours in the data, and compute if the density is high enough.
- The density can be obtained by dividing the number of neighbours by the area on the plot, i.e.

$$\text{Density} = \frac{\text{Number of neighbours}}{\pi\,(\text{Radius of the circular area occupied})^2}$$



Machine learning questions:

- How many anomalies are "visible"?
- Can we identify those anomalies?
- What do the anomalies signify?

10 Data visualisation

How to present data in the most engaging way?

- Is there a "story" hidden in your data?
- How to use visuals as information?
- How to tell the "story" effectively?

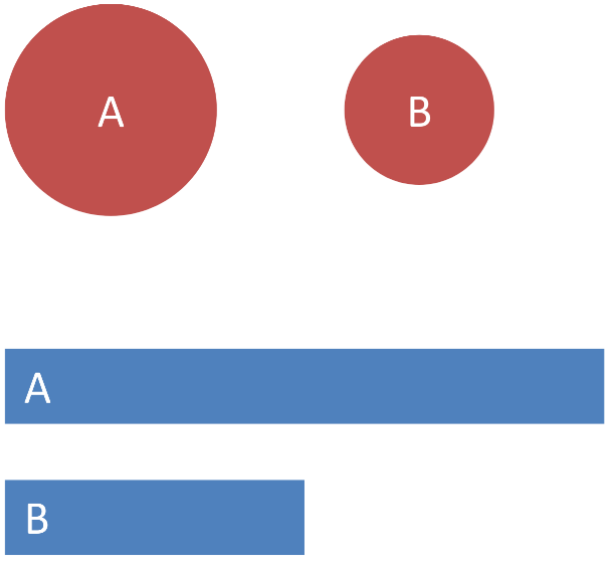
There are two goals when presenting data:

- Convey your story.
- Establish credibility.

10.1 Conveying your story

10.1.1 Effectiveness

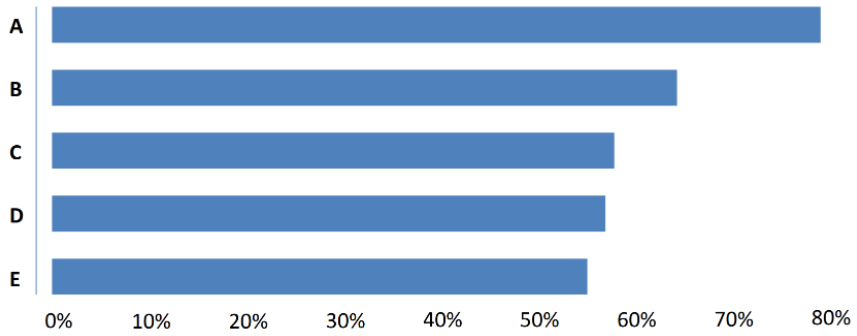
- A visualisation is more effective than another if the information conveyed by one visualisation is more **readily perceived** than the information in the other visualisation.
- Use encodings that people decode more quickly and accurately.



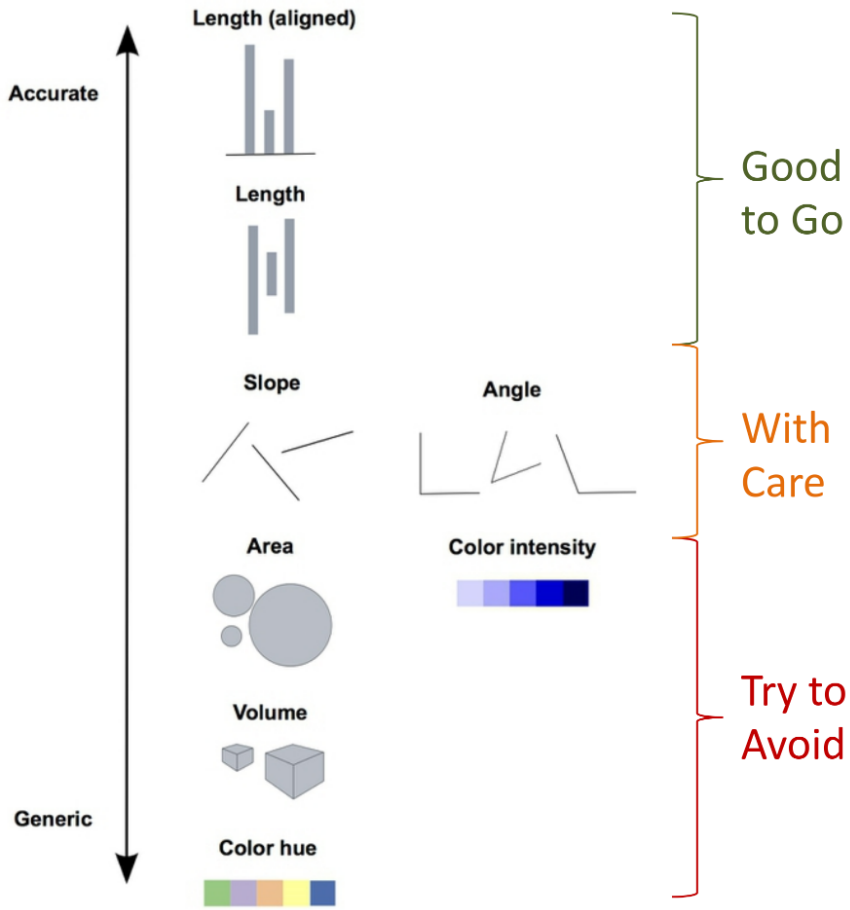
10.2 Establish credibility

10.2.1 Expressiveness

- A set of facts is expressible in a visual language if the visualisations in the language express **all the facts** in the set of data, and **only the facts** in the data.
- Tell the truth and nothing but the truth.
- Do not lie, and do not lie by omission.



10.3 Good versus bad visualisations



10.4 Data ink

Bold fonts and high contrast should be used to ensure that the data ink ratio is as high as possible.

10.4.1 Good example

2005 Sales Revenue (USD)

Sales Channel	Q1	Q2	Q3	Q4
Direct	383,383	403,939	437,373	538,583
Indirect	283,733	283,833	257,474	258,474
Total	667,116	687,772	694,847	797,057

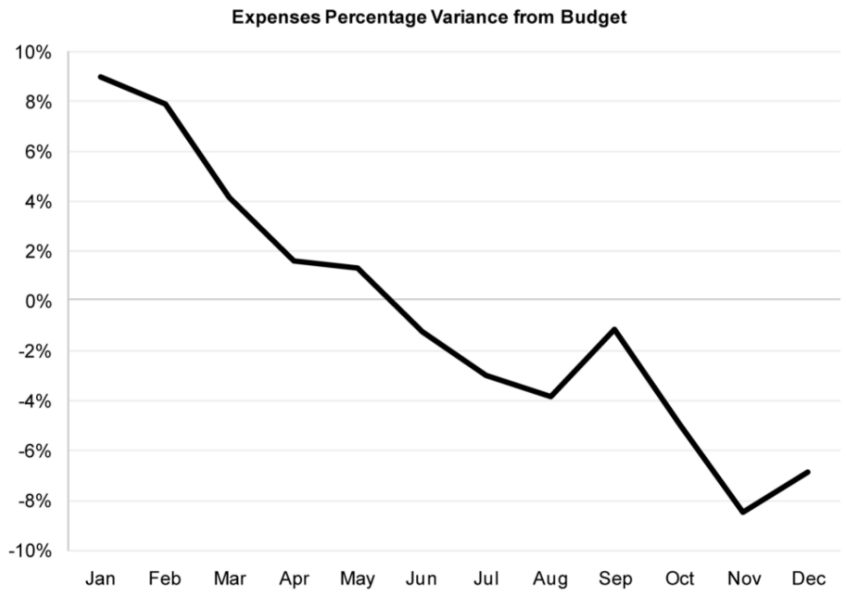
10.4.2 Bad example

2005 Sales Revenue (USD)

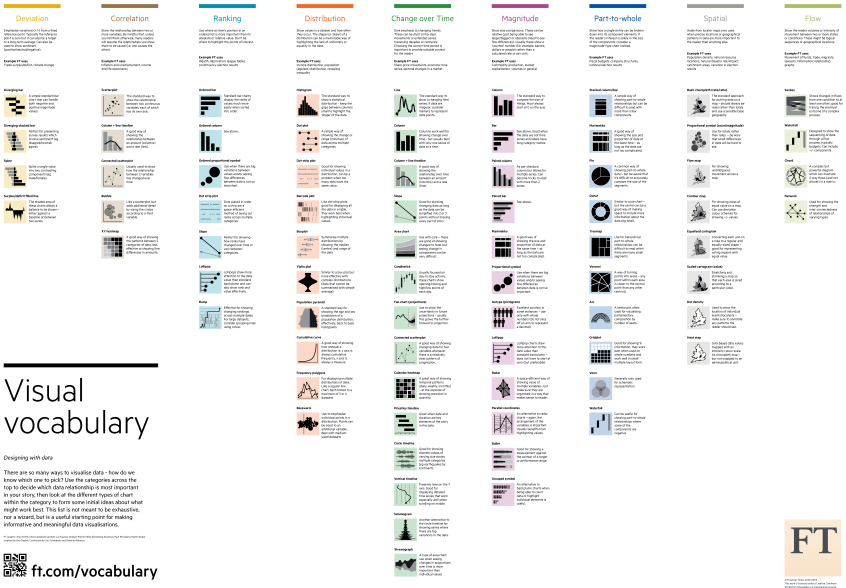
Sales Channel	Q1	Q2	Q3	Q4
Direct	383,383	403,939	437,373	538,583
Indirect	283,733	283,833	257,474	258,474
Total	667,116	687,772	694,847	797,057

10.5 Interpretation

Think about the context of your data and figure out the most effective way to present that data to your audience.



10.6 Visual vocabulary



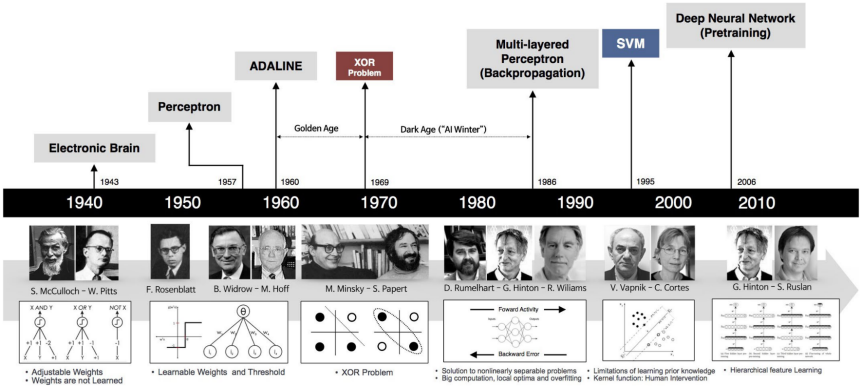
11 Introduction to artificial intelligence (AI)

AI is intelligence demonstrated by machines, in contrast to the *natural intelligence (NI) displayed by humans and other animals.

11.1 Founding fathers of AI

1. John MacCarthy
2. Marvin Minsky
3. Claude Shannon
4. Ray Solomonoff
5. Alan Newell
6. Herbert Simon
7. Arthur Samuel
8. Oliver Selfridge
9. Nathaniel Rochester
10. Trenchard More

11.2 Timeline



11.3 Views of AI

1. Thinking humanly
2. Acting humanly
3. Thinking rationally
4. Acting rationally

11.3.1 Thinking humanly: Cognitive Modelling

- In the 1960s, there was a "cognitive revolution" with the advent of information-processing psychology.
- It requires scientific theories of internal activities of the brain.
- Both approaches (cognitive science and cognitive neuroscience) are now distinct from AI.

11.3.2 Acting humanly: Turing test

- Alan Turing created the Turing test in the 1950s, initially called the imitation game.
- The aim is to test a machine's ability to exhibit intelligent behaviour equivalent to that of a human.
- In the test, a human evaluator judges a text transcript of a natural-language conversation between a human and a machine.
- The evaluator tries to identify the machine, and the machine passes if the evaluator cannot reliably tell them apart.
- Suggested major components of AI: knowledge, reasoning, language understanding, learning.

11.3.3 Thinking rationally: "Laws of Thought"

- Aristotle: What are correct arguments and thought processes?
  - Several Greek schools developed various forms of logic, which are notations and rules of derivation for thoughts and may or may not have proceeded to the idea of mechanisation.
- Direct line through mathematics and philosophy to modern AI.
- Problems:
  - Not all intelligent behaviour is mediated by logical deliberation.
  - What is the purpose of thinking? What thoughts should I have?

11.3.4 Acting rationally

- Rational behaviour is doing the right thing.
- The right thing is the thing that is expected to maximise goal achievement, given the available information.

11.4 AI beating humans

11.4.1 AlphaGo vs world Champions

March 9 to 15, 2016 (Lee Sedol)

- Time limit: 2 hours
- Venue: Seoul Four Seasons Hotel
- AlphaGo wins (4:1)

May 23 to 27, 2017 (Ke Jie)

- Venue: Wuzhen, China
- AlphaGo Wins (3:0)

11.4.2 Libratus vs world champions

- Libratus is the first AI to defeat top human poker players.
- The competition was held in 2017, from January 11 to 31, in Pittsburgh, with 120,000 hands being dealt.
- It has nothing to do with deep learning, but it represents algorithms for solving large-scale games.

11.5 Examples of natural language processing devices

- Mi Box Voice Assistant
- Google Assistant
- Amazon Alexa

11.6 Artificial intelligence for finance

11.7 What's next in AI for games?

- Stochastic, open environment
- Multiple players
- Sequential decision, online
- Strategic (selfish) behaviour
- Distributed optimisation

12 State of the art in data science and artificial intelligence

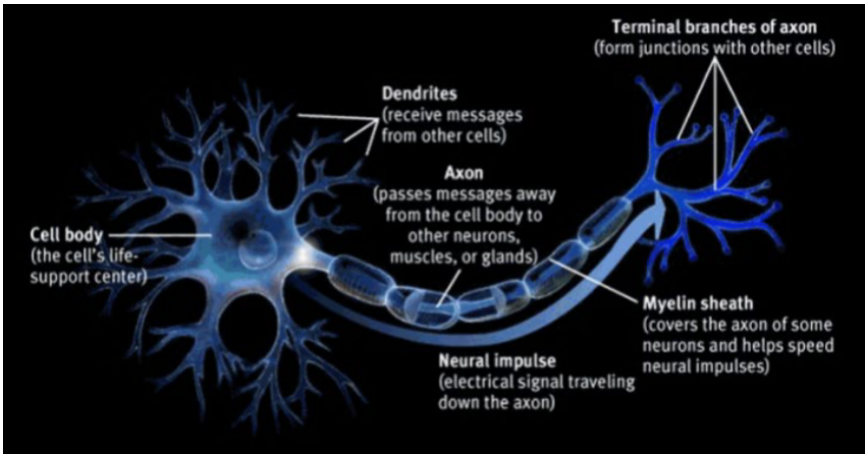
12.1 What is artificial intelligence?

- Artificial intelligence consists of 4 aspects:
  - Thinking
  - Perception
  - Action
  - Reasoning
- To create artificial intelligence is to build computer programs with algorithms that have the representation of the models targeted at thinking, perception, action and reasoning.

12.2 History of artificial intelligence

12.2.1 1943

Warren McCulloch and Walter Pitts created the first artificial neural model, based on the human neuron.

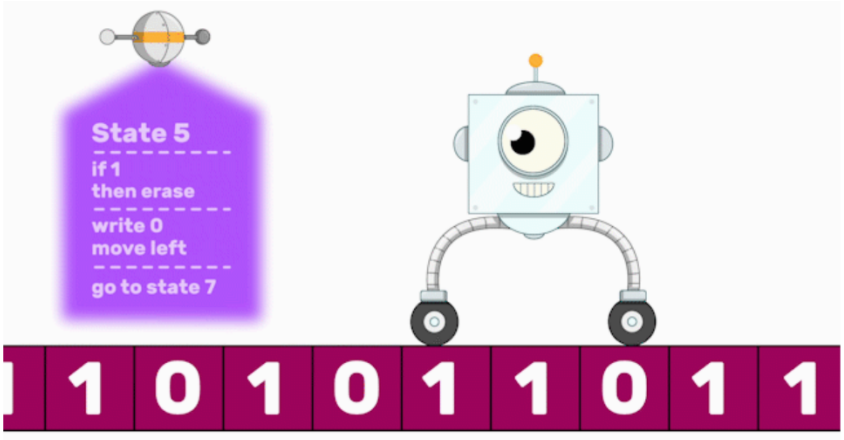


12.2.2 World War II (1939 - 1945)

During World War II, the Enigma machine was built to encrypt and decrypt communications.

12.2.3 1950

Alan Turing, the father of modern computer science, created the Turing machine.



12.2.4 1950s

Early AI programs include:

- Arthur Samuel's Checkers Program
- Newell and Simon's Logic Theorist
- Gelernter's Geometry Theorem Prover

12.2.5 1956 Darthmouth Conference

The 1956 Darthmouth Conference was attended by:

- Nathaniel Rochester
- Marvin Minsky
- John McCarthy
- Oliver Selfridge
- Ray Solomonoff
- Trenchard More
- Claude Shannon

12.2.6 1957

Frank Rosenblatt's created the perceptron, which was a device with the ability to learn.

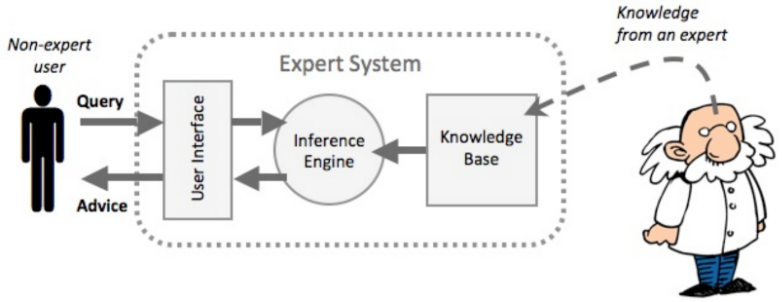
12.2.7 1961

The paper "Steps Toward Artificial Intelligence" was published by Marvin Minsky, talking about the possible uses of AI, such as:

- Search
- Pattern recognition
- Learning
- Planning
- Induction

12.2.8 The Golden Years (1960 - 1974)

- Robinson created a complete algorithm for logical reasoning.
- Appearance of expert systems, which are explicit, rule-based programs.



- Marvin Minsky's book, "Perceptrons", showed the limitations of simple neural networks in 1969.

12.2.9 The first AI winter (1974 - 1980)

The first AI winter was mainly due to:

- Low computational power.
- Results being primarily for toy problems.
- Loss of government funding in AI.

12.2.10 The second AI spring (1980 - 1988)

Japan created the Fifth Generation Computer Project (FGCP)

- It is a new generation of computers for "knowledge processing".
- They were expert systems used in businesses with specialised hardware.

Figure 1: Parallel Inference Machine in the 1980s

12.2.11 The second AI winter (1988 - 1993)

The second AI winter was due to:

- The goals of the Fifth Generation Computer Project being too ambitious.
- Marvin Minsky talking about the coming of the AI winter.
- Funding for AI ceasing.
- Personal computers becoming popular.

12.2.12 AI comeback (1994 - 2000)

- Resurgence of probability theories and focus on uncertainty.
- IBM's Deep Blue defeated Chess World Champion Garry Kasparov in chess.
- Moore's Law, which states that the number of transistors in an integrated circuit doubles every two years.

12.2.13 The quiet years (2001 - 2012)

- Internet boom.
- Focus on big data and statistical techniques.
- Graphics Processing Units (GPU) were created.
- AI is used for narrow use cases, such as:
  - 2005: Autonomous driving in the desert
  - 2011: IBM's Watson won Jeopardy
  - 2012: Convolutional Neural Networks (CNN) excelled at image recognition

12.2.14 2012

AlexNet wins the ImageNet competition, marking the resurgence of neural networks in computer vision.

12.2.15 2014

Ian Goodfellow introduces Generative Adversarial Networks (GANs), revolutionising image generation.

12.2.16 2015

Google DeepMind's AlphaGo beats the European Go champion, a milestone in game-playing AI.

12.2.17 2016

Google DeepMind introduces WaveNet, a deep generative model for generating raw audio waveforms.

12.2.18 2017

The paper "Attention Is All You Need" by Google introduces the transformer model, revolutionising natural language processing.

12.3 GPT and Large Language Models

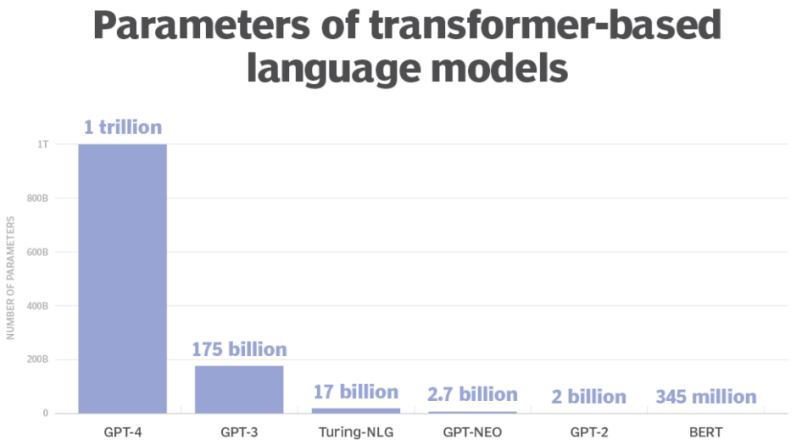
- AI has evolved from simple rule-based systems to complex models that understand and generate human language.
- Generative Pre-trained Transformer (GPT) models are a series of AI models designed by OpenAI, showcasing significant advancements in machine learning.
- Large Language Models (LLMs) like GPT are the cutting edge in natural language understanding and generation.

12.3.1 Examples

- Midjourney
- OpenAI's ChatGPT
- DeepSeek AI
- Dall-E 2
- Google's Gemini
- Microsoft and GitHub Copilot

12.3.2 What is GPT?

- GPT models use transformer architecture, enabling them to efficiently process and generate language.
- These models are "pre-trained" on vast amounts of text data, allowing them to understand context and produce relevant responses.
- GPT's ability to generate coherent and contextually appropriate text has revolutionised AI's interaction with human language.
- GPT 1 has 117 million parameters, GPT 2 has 1.5 billion parameters, and GPT 3 has 175 billion parameters.



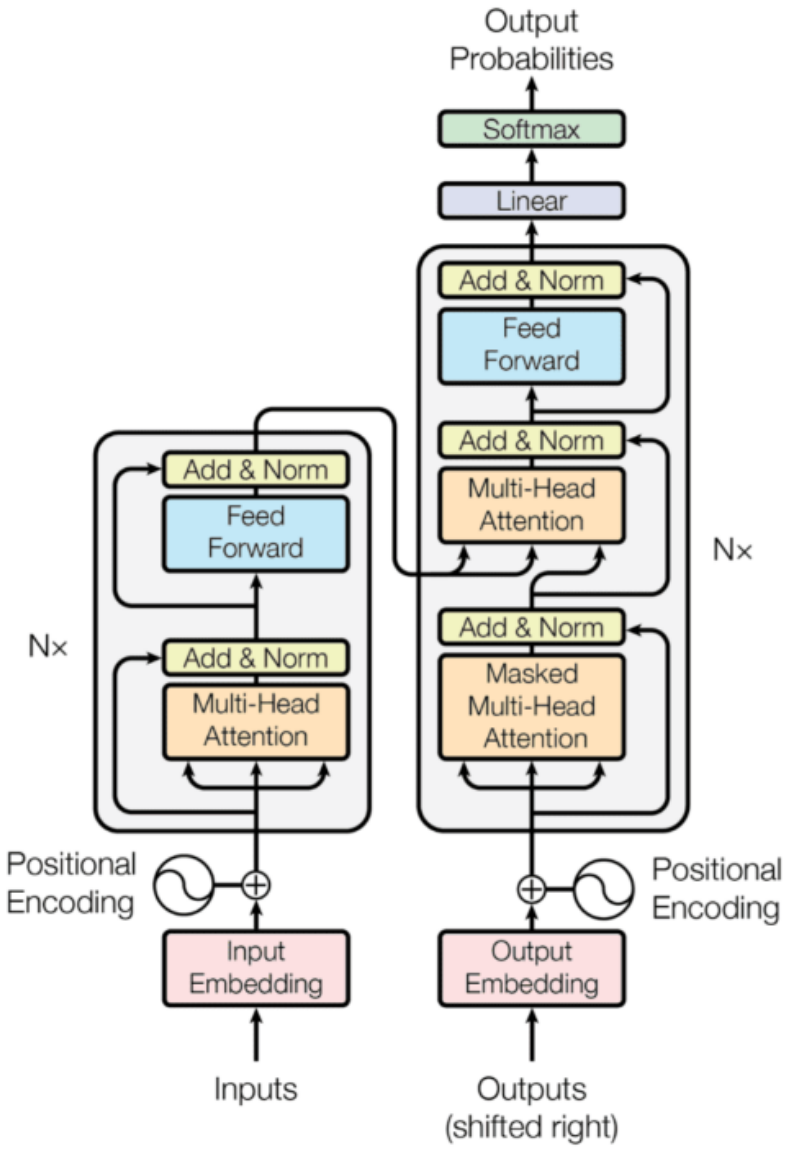


Figure 2: The encoder-decoder structure of the Transformer architecture.

12.3.3 Limitations

- Data bias, as GPT models can inherit biases present in their training data.
- Computation cost, as training large models requires significant computation resources.
- Ethical concerns, as there is potential for misuse in misinformation, privacy, and security.

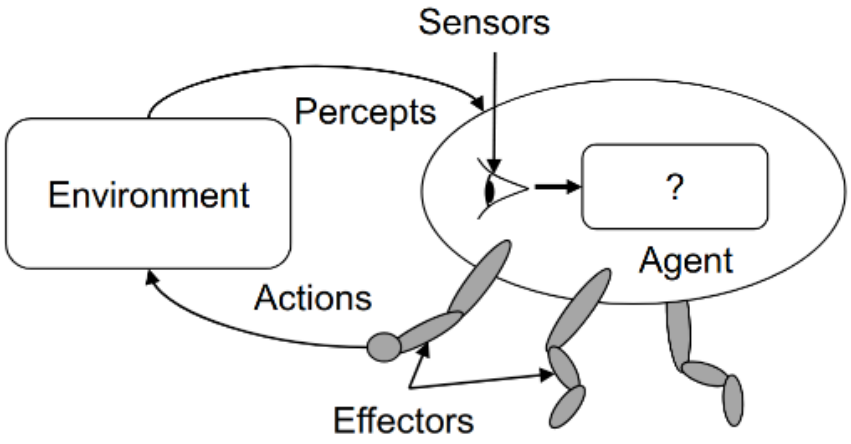
12.4 Features required for a machine to pass the Turing test

The features required for a machine to pass the Turing test are:

- Natural language processing (NLP), as NLP is required to communicate with the interrogator in general human language like English
- Knowledge representation, as it is needed to store and retrieve information during the test.
- Automated reasoning, which is needed to use the previously stored information for answering the questions.
- Machine learning, which is needed to adapt new changes and can detect generalised patterns.
- Vision (for total Turing test), which is needed to recognise the interrogator actions and other objects during a test.
- Motor control (for total Turing test), which is needed to act upon objects if requested.

13 Intelligent agents

An **agent** is an entity that **perceives** through sensors, like eyes, ears, cameras, and infrared range sensors, and **acts** through effectors, such as hands, legs and motors.



13.1 Rational agents

- A rational agent is one that does the **right** thing.

The rationality of the agent depends on:

- Performance measures.
- Everything that the agent has perceived so far.
- Built-in knowledge about the environment.
- Actions that can be performed.

13.1.1 Example: Google's X2 Driverless Taxi

- **Percepts**: Video, speed acceleration, engine status, GPS, radar, etc.
- **Actions**: Steer, accelerate, brake, horn, display, etc.
- **Goals**: Safety, arrive at destination, maximise profits, obey laws, passenger comfort, etc.
- **Environment**: Singapore urban streets, highways, traffic, pedestrians, weather, customers, etc.

13.1.2 Example: Medical diagnosis system

- **Percepts**: Symptoms, findings, patient's answers, etc.
- **Actions**: Questions, medical tests, treatments, etc.
- **Goals**: Healthy patient, faster recovery, minimise costs, etc.
- **Environment**: Patient, hospital, clinic, etc.

13.2 Autonomous agents

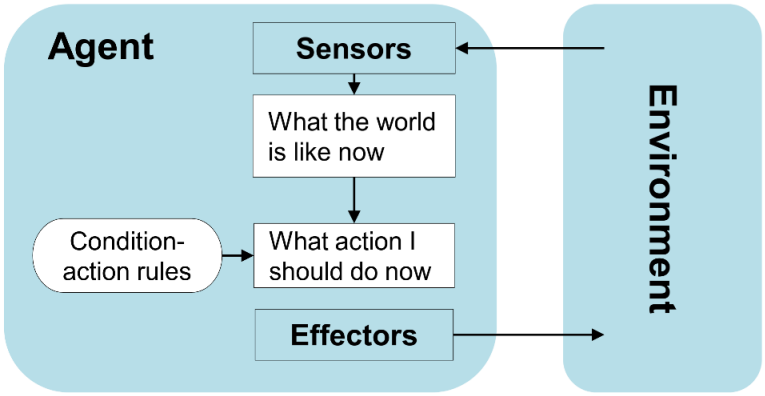
- Autonomous agents do **not** rely entirely on the built-in knowledge about the environment, i.e. they are not entirely pre-programmed.
- Otherwise, the agent will only operate successfully when the built-in knowledge is all correct.
- Such agents adapt to the environment through experience.

13.2.1 Example: Driverless car

- Learning to drive in a driving centre.
- Drive at NTU.
- Drive on public roads.
- Drive on highways.
- Drive in City Hall.

13.3 Simple reflex agents

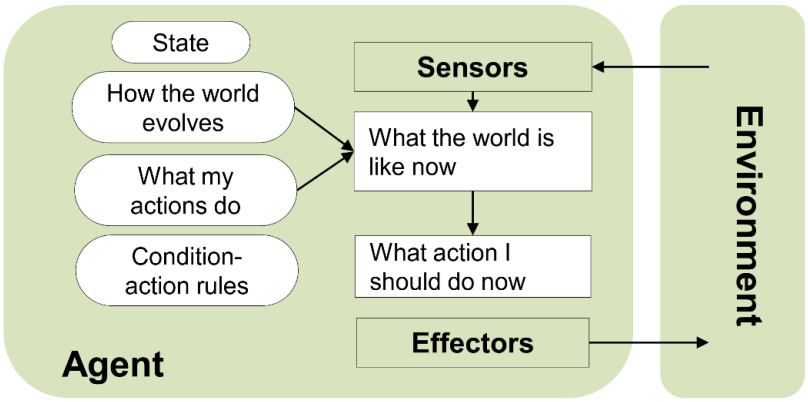
1. Find the **rule** whose condition matches the current situation, as defined by the percept.
2. Perform the action associated with that rule.



For example, if the car in front is braking, then initiate braking.

13.4 Reflex agent with state

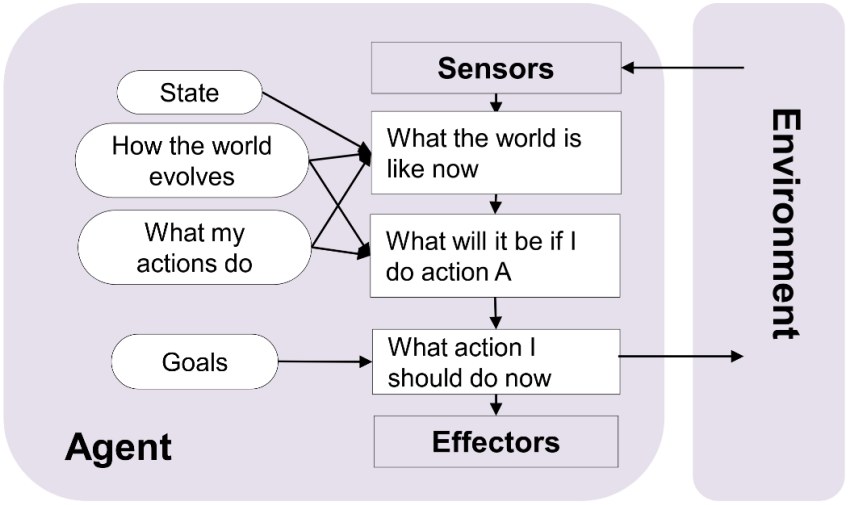
1. Find the rule whose condition matches the current situation, as defined by the percept and the stored internal state.
2. Perform the action associated with that rule.



For example, if the agent was at NTU yesterday, and there is no traffic jam currently, then go to Orchard.

13.5 Goal-based agents

Goal-based agents need some sort of goal information.

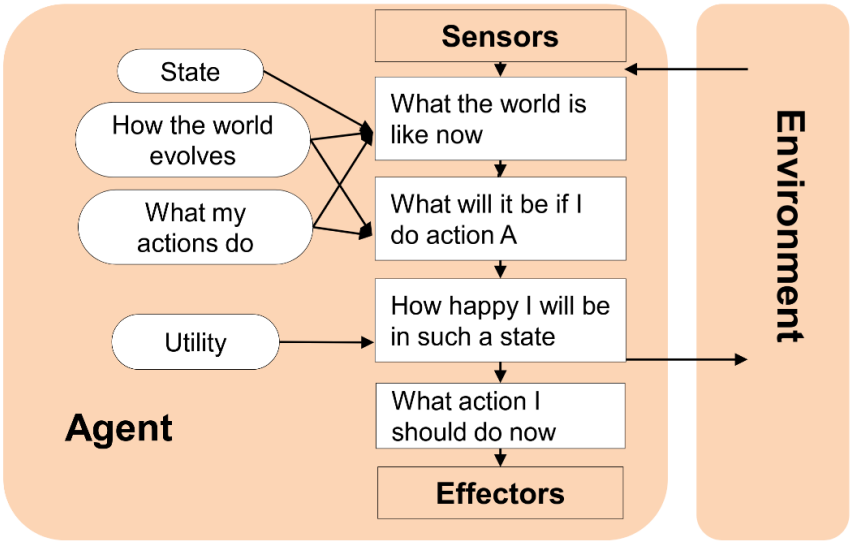


For example, for a driverless taxi:

- At a junction, which is a known state, should I go left, right or straight on?
- Do I reach my destination, which is Orchard?

13.6 Utility-based agents

- There may be many action sequences that can achieve the same goal, which action sequence should it take?
- How happy the agent will be if it attains a certain state is called the utility.



For example, for a driverless taxi:

- Go to Orchard (destination) via PIE or AYE?
- Which one charges a lower fare?

13.7 Types of environments

- Accessible vs inaccessible: Agent's sensory apparatus gives it access to the complete state of the environment.
- Deterministic vs nondeterministic: The next state of the environment is completely determined by the current state and the actions selected by the agent.
- Episodic vs sequential: Each episode is not affected by the previous taken actions.
- Static vs dynamic: The environment does not change while an agent is deliberating.
- Discrete vs continuous: A limited number of distinct percepts and actions.

13.7.1 Example: Driverless Taxi

- Accessible? No. Some traffic information on the road is missing.
- Deterministic? No. Some cars in front may turn right suddenly.
- Episodic? No. The current action is based on the previous driving actions.
- Static? No. When the taxi moves, other cars are moving as well.
- Discrete? No. Speed, distance and fuel consumption are in real domains.

13.7.2 Example: Chess

- Accessible? Yes. All positions in a chessboard can be observed.
- Deterministic? Yes. The outcome of each movement can be determined.
- Episodic? No. The action depends on previous movements.
- Static? Yes. When there is no clock, when you are considering the next step, your opponent can't move. Semi-static in the case where there is a clock, as you give up the movement once the time is up.
- Discrete? Yes. All positions and movements are in discrete domains.

13.7.3 Example: Minesweeper

- Accessible? No. Mines are hidden.
- Deterministic? No Mines are randomly assigned in different positions.
- Episodic? No. The action is based on previous outcomes.
- Static? Yes. When you are considering the next step, the environment does not change.
- Discrete? Yes. All positions and movements are in discrete domains.

13.7.4 Example: Slot machines

- Accessible? No.
- Deterministic? No.
- Episodic? Yes.
- Static? Yes.

13.8 Design of a problem-solving agent

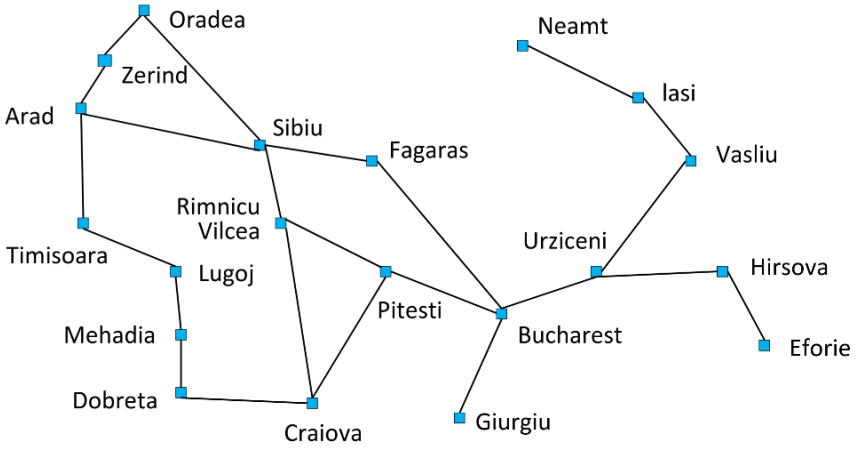
13.8.1 Idea

- A problem-solving agent systematically considers the expected outcomes of different possible sequences of actions that lead to states of known value.
- Choose the best one.
  - Shortest journey from A to B?
  - Most cost-effective journey from A to B?

13.8.2 Steps

1. Goal formulation
2. Problem formulation
3. Search process
   - Uninformed search for a search without prior knowledge.
   - Informed search for a search with prior knowledge.
4. Action execution, which is to follow the recommended route.

13.8.3 Example: Goal based agent



On a holiday in Romania:

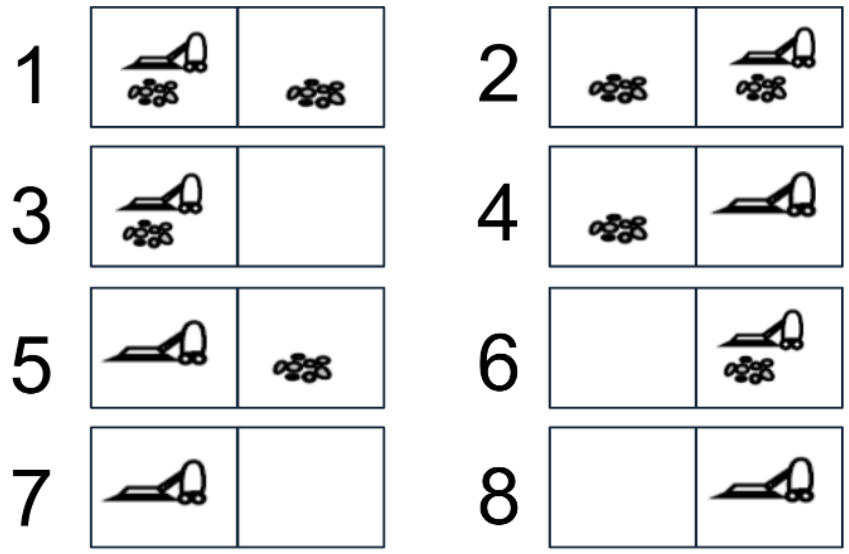
- Initial state: Currently in Arad. The flight leaves tomorrow from Bucharest.
- Goal: To be in Bucharest. Other factors include cost, time, most scenic route, etc.
- State: To be in a city, which is defined by the map above.
- Action: Transition between states, which are the highways defined by the map.

13.8.4 Example: Vacuum cleaner agent

- Robotic vacuum cleaners move autonomously.
- Some can come back to a docking station to charge their batteries.
- A few are able to empty their dust containers into the dock as well.

13.8.5 Example: A simple vacuum world

Two locations, each location may or may not contain dirt, and the agent may be in one location or the other.



- 8 possible world states.
- Possible actions include turn left, turn right, and suck.
- The goal is to clean up all dirt. There are two goal states, in the diagram above, 7 and 8.

13.9 Well-defined formulation

13.9.1 Definition of a problem

The information used by an agent to decide what to do.

13.9.2 Specification

- Initial state.
- Action set, which is all the available actions (successor functions).
- State space, which is all the states reachable from the initial state.
  - The solution path is a sequence of actions from one state to another.
- Goal test predicate, which can either be a single state, an enumerated list of states, or abstract properties.
- Cost function, like a path cost $g(n)$ which is the sum of all action step costs along the path.

13.9.3 Solution

The solution is a path, or a sequence of operators leading from the initial state to a state that satisfies the goal test.

13.10 Measuring problem-solving performance

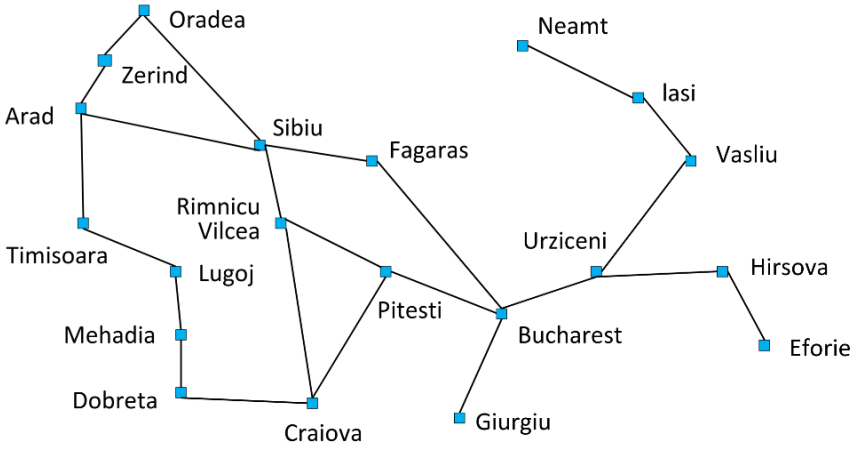
Search cost:

- What does it cost to find the solution? How long (time)? How many resources were used (memory)?

Total cost of problem-solving:

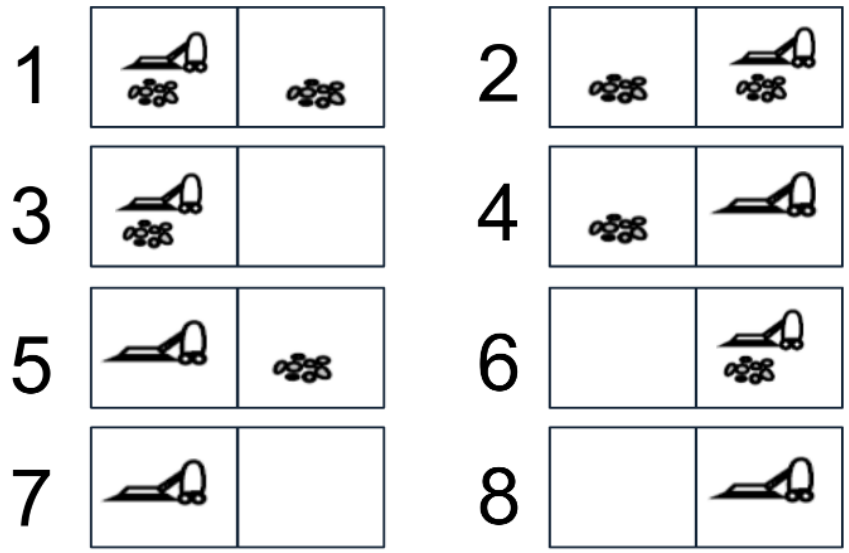
- Search cost ("offline", as in when the agent is finding a solution) + execution cost ("online", as in when the agent is executing the solution)
- Trade-offs are often required. For example, you can either search for a very long time for the optimal solution, or search for a shorter time for a "good enough" solution.

13.10.1 Single-state problem example

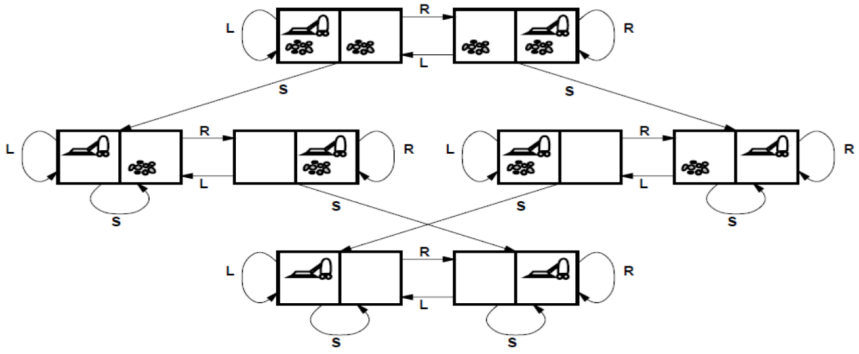


- Initial state: In Arad.
- Set of possible action and the corresponding next states:
  - Arad → Zerind
  - Arad → Timisoara
  - Arad → Sibiu
- Goal test, which is explicit, like the state being equal to being in Bucharest.
- Path cost, which is a function. It can be the sum of distances or the number of operators in the executed solution. The executed solution is a sequence of operators leading from the initial state to a goal state.

13.10.2 Example: Vacuum world (single-state version)

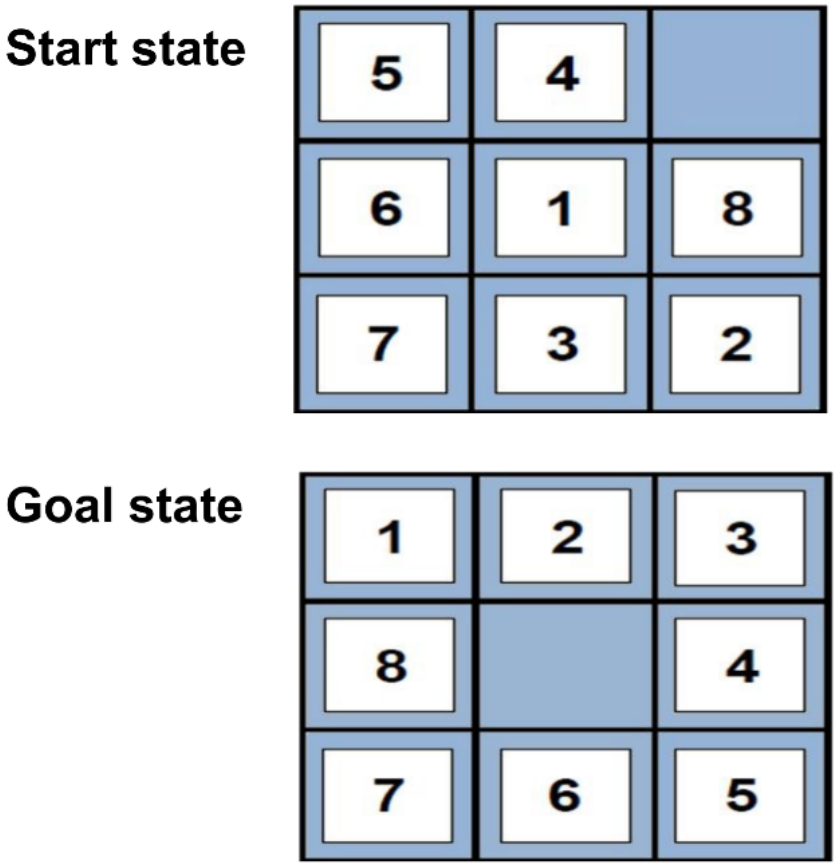


- Initial state: One of the 8 states shown above.
- Actions: Turn left, turn right, and suck.
- Goal test: No dirt in any square.
- Path cost: 1 per action.



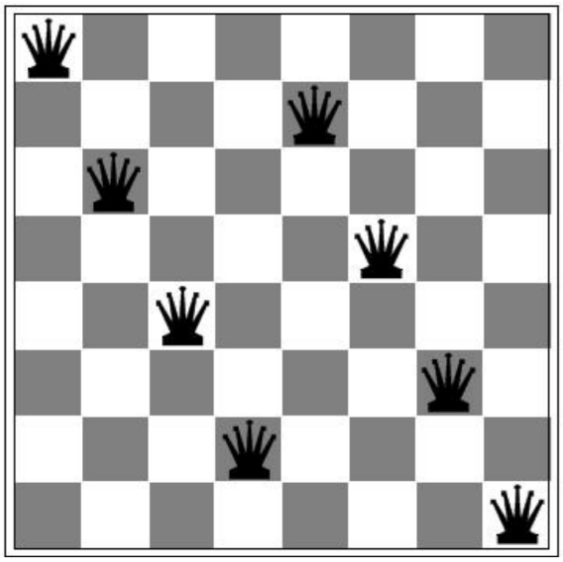
13.10.3 Example: 8-puzzle

- States: integer locations of tiles.
  - Number of states = 9!
- Actions: Move blank left, right, up or down.
- Goal test: Equals to goal state or not, which is shown in the picture below.
- Path cost: 1 per move.



13.10.4 Example: 8-queens

- States: Any arrangement of 0 to 8 queens on the board.
- Actions: Add a queen to any empty square.
- Goal test: 8 queens are on the board, none under attack.
- Path cost: Not necessary.



13.10.5 Real world problems

Route finding problems

- Routing in computer networks
- Robot navigation
- Automated travel advisory
- Airline travel planning

Touring problems:

- The Travelling salesman problem is a type of shortest tour problem. The goal is to find the shortest path that visits every city exactly once and returns to the origin city.

14 Search algorithms

- Search algorithms explore the state space by generating successors of already-explored states.
- The frontier is the candidate nodes for expansion in the explored set.

14.1 Search strategies

- A search strategy is defined by picking the order of node expansion.
- Strategies are evaluated along the following dimensions:
  - Completeness: Does it always find a solution if one exists?
  - Time complexity: How long does it take to find a solution? This is also the number of nodes generated.
  - Space complexity: The maximum number of nodes in memory.
  - Optimality: Does it always find the best (least-cost) solution?

14.1.1 Branching factor

The branching factor is the maximum number of successors of any node. It is also called the average branching factor.

14.1.2 Uninformed search

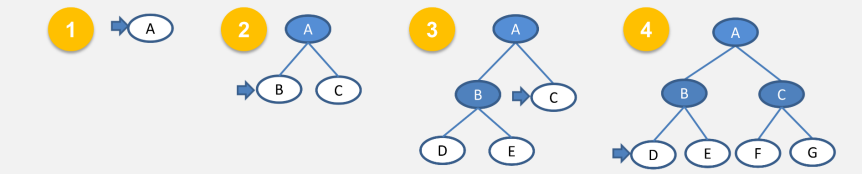
- Uninformed search strategies use only the information available in the problem definition.
- Some examples include:
  1. Breadth-first search
  2. Uniform-cost search
  3. Depth-first search
  4. Depth-limited search
  5. Iterative deepening search

14.1.3 Informed search

- Informed search strategies use problem-specific knowledge to guide the search.
- They are usually more efficient.

14.2 Breadth-first search (BFS)

Breadth-first search expands the **shallowest** unexpanded node, which can be implemented by a first-in-first-out (FIFO) queue.



Where:

- b is the maximum branching factor of the search tree
- d is the depth of the least-cost solution

The algorithm is complete, and optimal when all step costs are equal.

14.2.1 Complexity

- Consider a hypothetical state-space, where every node can be expanded into b new nodes, with a solution of path-length d.
- The time complexity would be:

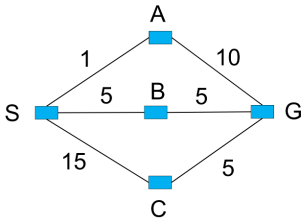
$$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

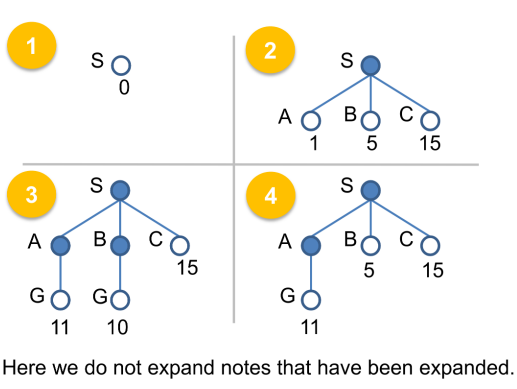
- The space complexity would be the same as the time complexity, as breadth-first search keeps every node in memory, i.e. $O(b^d)$.

Depth	Nodes		Time		Memory
0	1	1	millisecond	100	bytes
2	111	0.1	seconds	11	kilobytes
4	11111	11	seconds	1	kilobytes
6	10^6	18	minutes	111	megabyte
8	10^8	31	hours	11	gigabytes
10	10^{10}	128	days	1	terabyte
12	10^{12}	35	years	111	terabytes
14	10^{14}	3500	years	11111	terabytes

14.3 Uniform-cost search (UCS)

- Uniform-cost search considers **edge costs**, and hence expands the unexpanded node with the **least** path cost g.
- It is a modification of breadth-first search.
- Instead of using a first-in-first-out (FIFO) queue, use a priority queue with path cost $g(n)$ to order the elements.
- Breadth-first search = uniform-cost search with $g(n) = \text{Depth}(n)$





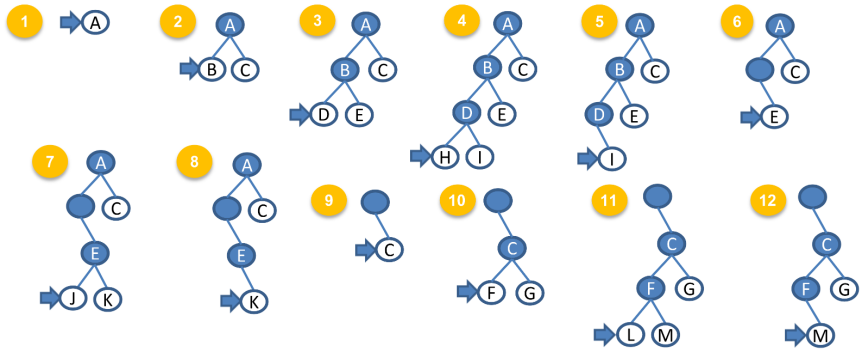
Here we do not expand notes that have been expanded.

14.3.1 Characteristics

- Uniform-cost search is complete.
- The time complexity of uniform-cost search is the number of nodes with the path cost g that is less than or equal to the cost of the optimal solution. It is equivalent to the number of nodes that pop out from the priority queue.
- The space complexity of uniform-cost search is the same as the time complexity, i.e. it is the number of nodes with the path cost g that is less than or equal to the cost of the optimal solution.
- Uniform-cost search is also optimal.

14.4 Depth-first search (DFS)

- Depth-first search expands the deepest unexpanded node, which can be implemented by using a last-in-first-out (LIFO) stack.
- It backtracks only when there is no more expansion.



14.4.1 Characteristics

Let m be the maximum depth of the state space.

- For the characteristic of completeness:
  - Depth-first search is not complete for infinite-depth spaces.
  - It is not complete for finite-depth spaces with loops, but it is complete when these finite-depth spaces with loops have repeated-state checking.
  - It is complete for finite-depth spaces without loops.
- The time complexity of depth-first search is $O(b^m)$. If the solutions are dense, depth-first search may be much faster than breath-first search.
- The space complexity of depth-first search is $O(bm)$.
- Depth-first search is not optimal.

14.4.2 Depth-limited search

- Depth-limited search is a depth-first search with a **cut-off** on the maximum depth (I) of a path to avoid infinite searching.
- It is complete, if the maximum depth (I) is greater or equal to the depth of the least cost solution.
- The time complexity of depth-limited search is $O(b^l)$.
- The space complexity of depth-limited search is $O(bl)$.
- Depth-limited search is not optimal.

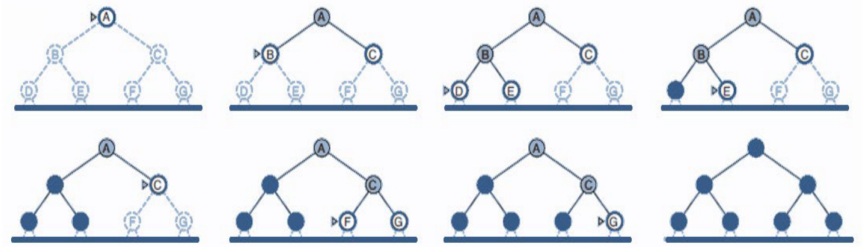
14.5 Iterative deepening search

Iterative deepening search iteratively estimates the maximum depth I of a depth-limited search one by one.

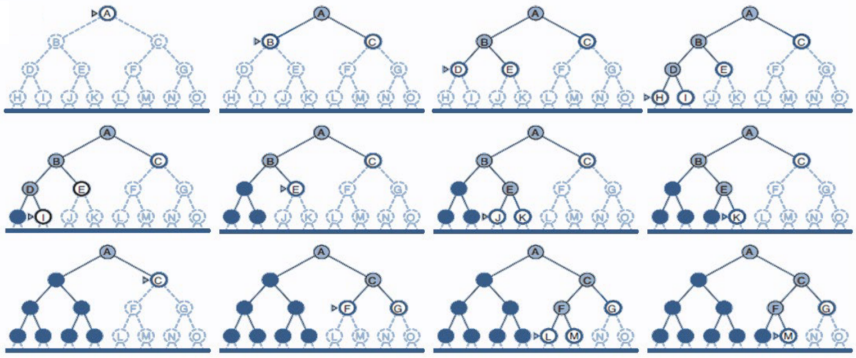
14.5.1 Limit is 0

14.5.2 Limit is 1

14.5.3 Limit is 2



14.5.4 Limit is 3



14.5.5 Pseudocode implementation

```
function interative_deepening_search(problem) returns a solution sequence
    inputs: problems, a problem

    for depth = 0 to infinity do
        if depth_limited_search(problem, depth) succeeds then return its result end
    end
end
```

14.5.6 Characteristics

- The iterative deepening search is complete.
- The time complexity of the iterative deepening search is $O(b^d)$.
- The space complexity of the iterative deepening search is $O(bd)$.
- Iterative deepening search is optimal.

14.6 Summary of search algorithms

Criterion	Breadth-first	Uniform-cost	Depth-first	Depth-limited	Iterative deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{\frac{d}{2}}$
Space	b^d	b^d	bm	bl	bd	$b^{\frac{d}{2}}$
Optimal	Yes	Yes	No	No	Yes	Yes
Complete	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

14.7 General search

14.7.1 Uninformed search strategies

- **Systematic** generation of new states to pass the goal test
- **Inefficient**, as it has exponential space and time complexity.

14.7.2 Informed search strategies

- Uses **problem-specific** knowledge to decide the order or node expansion.
- For example, the best-first search expands the most desirable unexpanded node. It makes use of an **evaluation function** to **estimate** the **"desirability"** of each node.

14.8 Evaluation function

- The path-cost function $g(n)$ is the cost from the initial state to the current state (search node) n. It does not provide any information on the cost **towards the goal**.
- The evaluation function needs to estimate costs to the closest goal.
- The "heuristic" function $h(n)$ is the estimated cost of the cheapest path from n to a goal state $h(n)$.
  - The exact cost cannot be determined.
  - The function depends only on the state at that node.
  - $h(n)$ is not larger than the real cost.

14.9 Greedy search

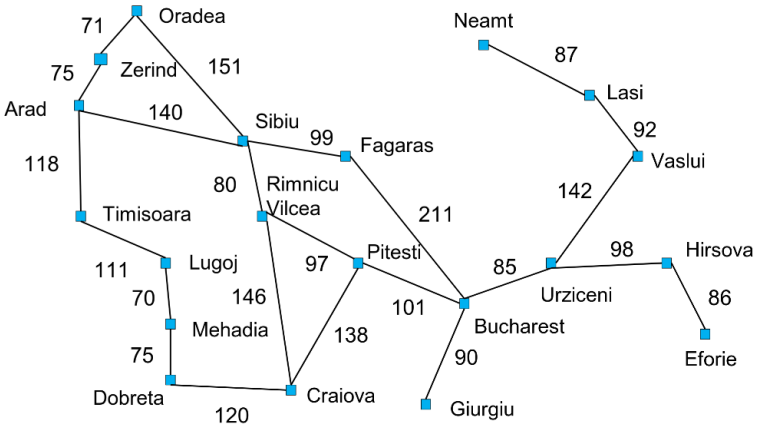
- Greedy search expands the node that **appears** to be closest to the goal.
- The evaluation function $h(n)$ estimates the cost from n to the **goal**.

14.9.1 Pseudocode implementation

```
function greedy_search(problem) returns solution
    return best_first_search(problem, h)    -- h (goal) = 0
end
```

14.9.2 Example

$h(n)$ is the straight-line distance from n to **Bucharest**.

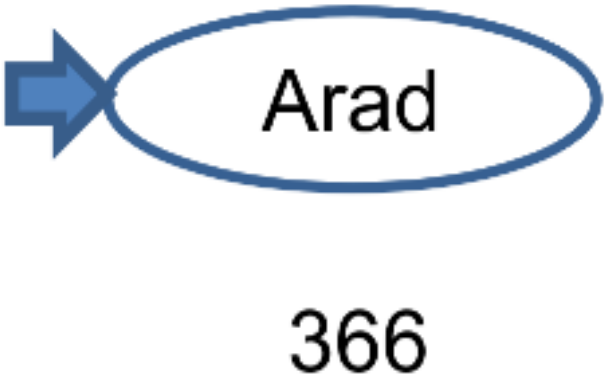


Straight-line distance to Bucharest:

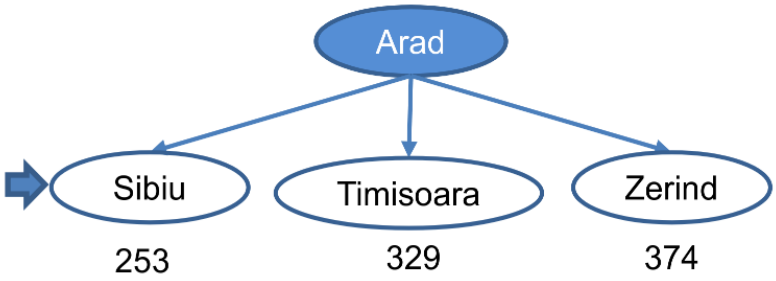
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

The heuristic of the straight-line distance to Bucharest is useful, but potentially fallible. Heuristic functions are problem-specific as well.

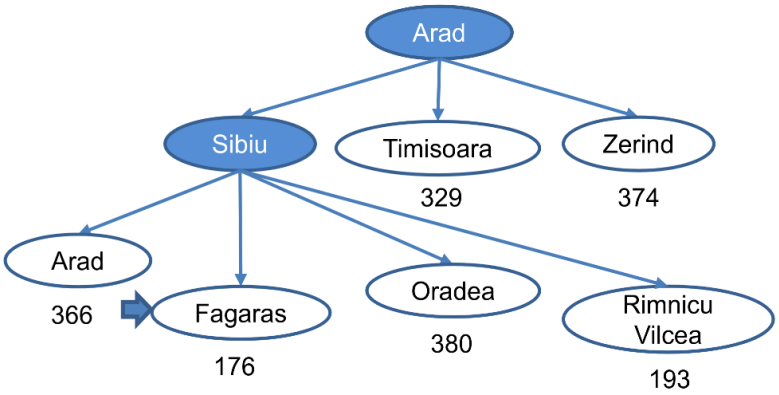
1. Initial state:



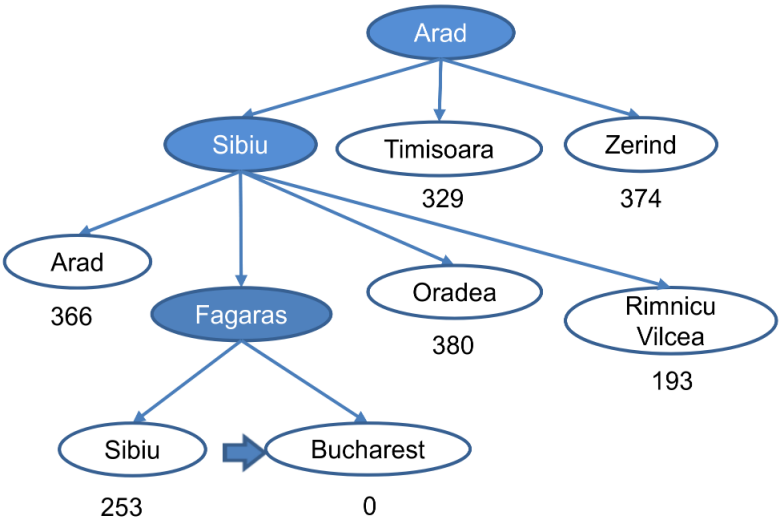
2. After expanding Arad:



3. After expanding Sibiu:



4. After expanding Fagaras:



14.9.3 Characteristics

Let m be the maximum depth of the search space.

- The greedy search algorithm is not complete.
- The time complexity of greedy search is $O(b^m)$.
- The space complexity of greedy search is $O(b^m)$ as it keeps all nodes in memory.
- The greedy search algorithm is not optimal.

14.10 A* search

- Uniform-cost search
  - $g(n)$ is the cost to reach n (past experience).
  - It is optimal and complete, but can be very inefficient.
- Greedy search
  - $h(n)$ is the cost from n to the goal (future prediction).
  - It is neither optimal nor complete, but it cuts search space considerably.
- A* search combines greedy search with uniform-cost search.
- Evaluation function:

$$f(n) = g(n) + h(n)$$

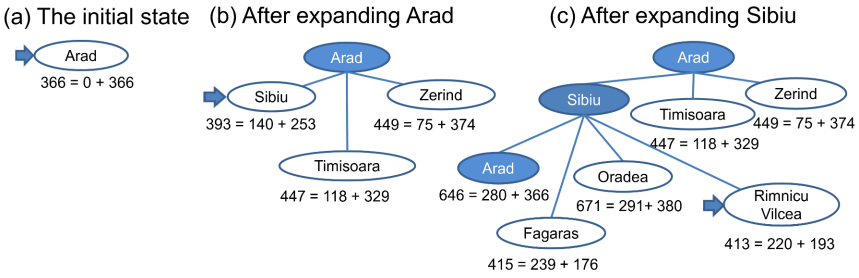
  - $f(n)$ is the estimated **total** cost of the path through n to the goal
  - If $g = 0$, then it becomes a greedy search.
  - If $h = 0$, then it becomes a uniform-cost search.

14.10.1 Pseudocode implementation

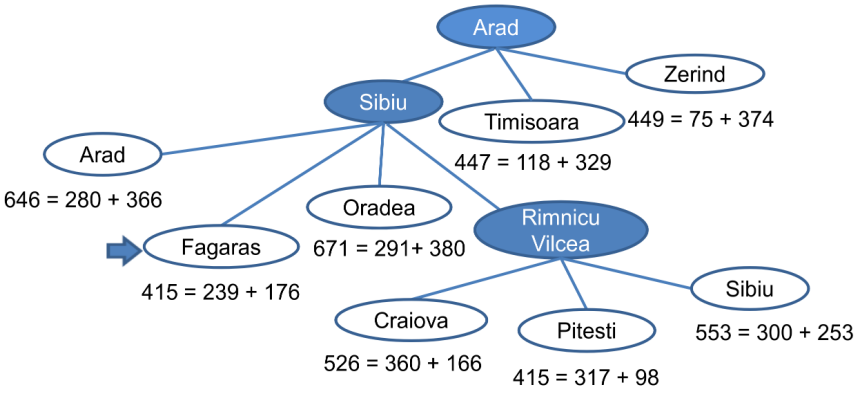
```
function a_star_search(problem) returns solution
    return best_first_search(problem, g + h)
end
```

14.10.2 Example

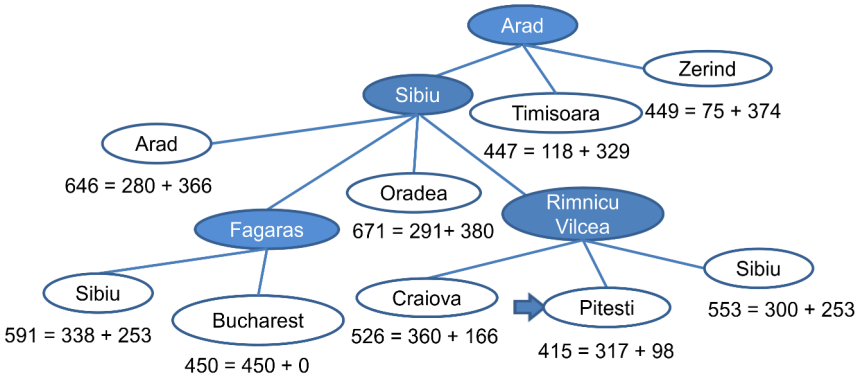
Using best-first-search with evaluation function $g + h$:



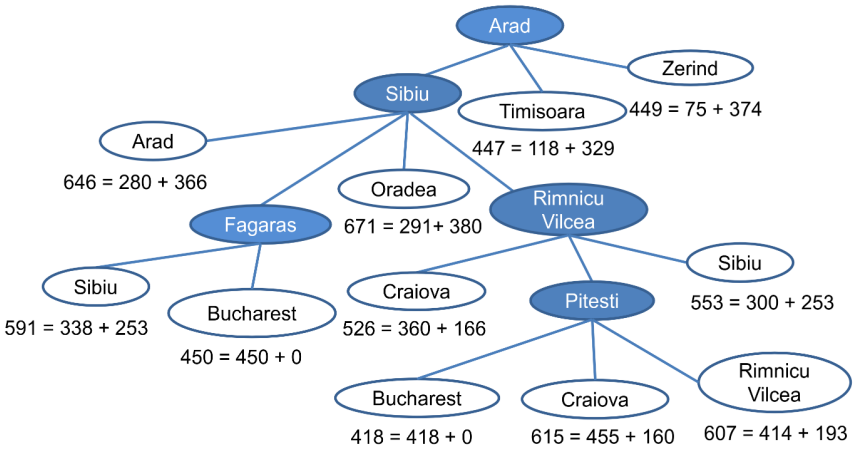
(d) After expanding Rimnicu Vilcea:



(e) After expanding Fagaras:



(f) After expanding Pitesti:

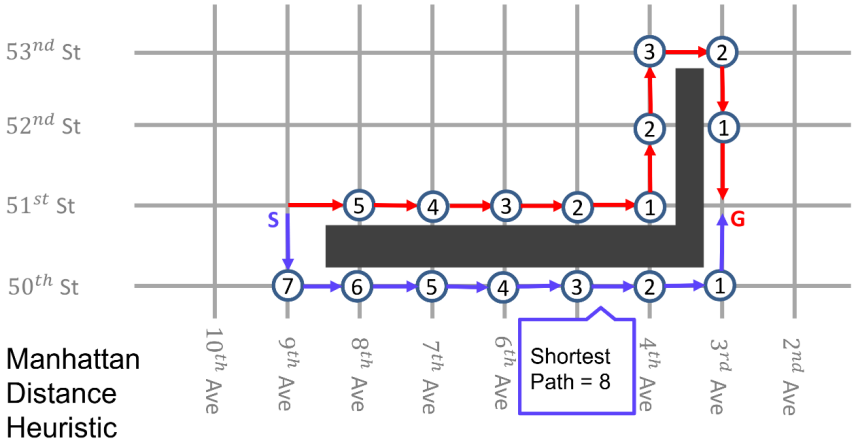


14.10.3 Complexity

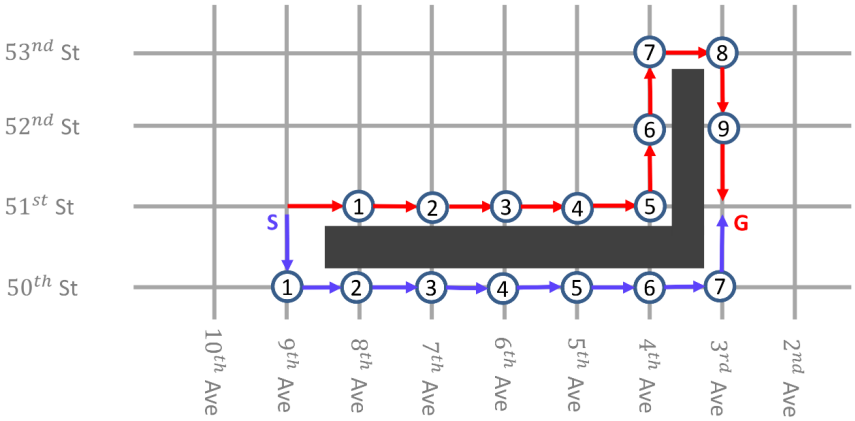
- The time complexity of A* search is exponential with respect to the length of the solution.
- The space complexity of A* search is the same as the time complexity, i.e. it is exponential with respect to the length of the solution.
- However, with a good heuristic, significant savings are still possible compared to uninformed search methods.

14.11 Route finding in Manhattan example

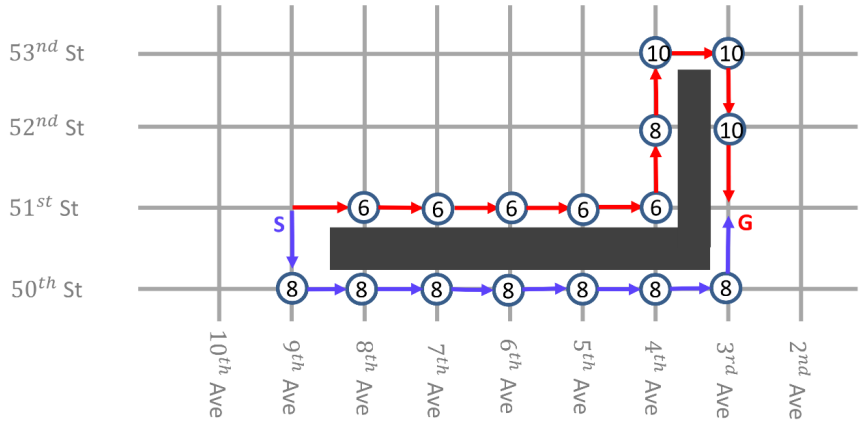
14.11.1 Greedy search



14.11.2 Uniform-cost search



14.11.3 A* search



15 Constraint satisfaction

The goal of a constraint satisfaction is to discover some state that satisfies a given set of constraints. Some examples include:

- 8-queens problem
- Crypt arithmetic puzzle
- Sudoku
- Minesweeper

15.1 Definitions

15.1.1 State

A state of the problem is defined by an **assignment** of values to some or all of the variables.

15.1.2 Consistent assignment (Legal assignment)

An assignment that does not violate any constraints.

15.1.3 Solution

A **solution** to a constraint satisfaction problem is an assignment that gives every variable a value (**complete**), and the assignment satisfies all the constraints.

15.2 Real world problems

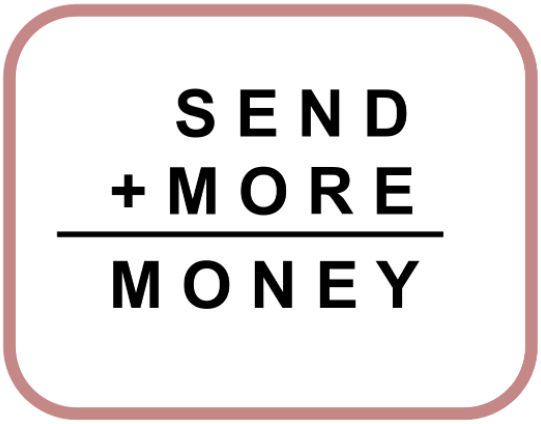
- Assignment problems, such as who teaches what class
- Timetabling problems, such as which class is offered when and where
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floor-planning

15.3 Constraint satisfaction problem (CSP)

- The state is defined by the **variables** V_i with values from the domain D_i.
- For the 8-queens example:
  - The variables would be the locations of each of the eight queens.
  - The values would be the squares on the board.
- The goal test is a set of **constraints** specifying allowable combinations of values for subsets of variables.
  - For the 8-queens example, the goal test would be to have no two queens in the same row, column or diagonal.

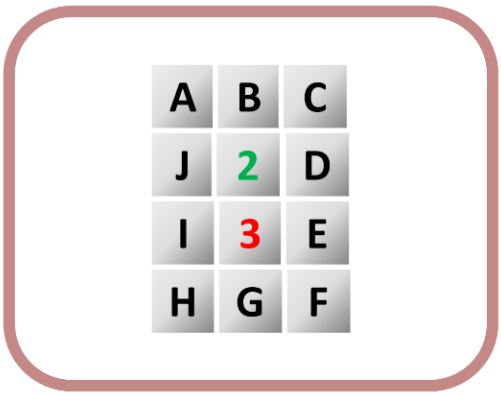
15.4 Examples

15.4.1 Crypt arithmetic puzzle



- Variables: D, E, M, N, O, R, S, Y
- Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
  - $Y = D + E$ or $Y = D + E - 10$, etc.
  - $D \neq E, D \neq M, D \neq N$, etc.
  - $M \neq 0, S \neq 0$. These are unary constraints, which concern the value of a single variable.

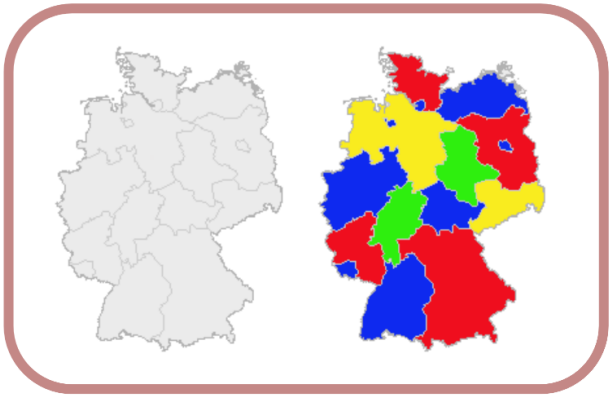
15.4.2 Minesweeper



- Variables: The cells
- Domains: $\{0; 1\}$ representing $\{\text{safe}, \text{mined}\}$
- Constraints: Each cell has a number $m \in \{1, \ldots, 8\}$ indicating the number of mines nearby, so m is equal to the sum of the value of neighbouring cells.

15.4.3 Map colouring

Colour a map so that no adjacent parts have the same colour.



- Variables: Countries Ci
- Domains: $\{\text{Red}, \text{Blue}, \text{Green}\}$
- Constraints: $C1 \neq C2, C1 \neq C5$, etc, which are binary constraints.

15.5 Applying standard search

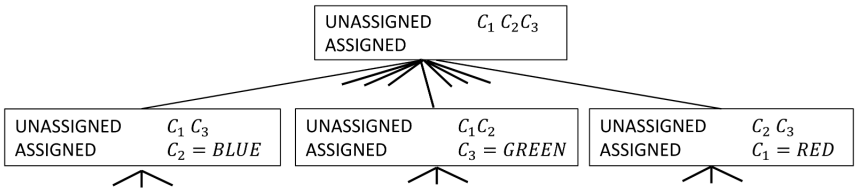
- The states are defined by the values assigned so far.
- The initial state is to have all variables unassigned.
- The possible actions are to assign a value to an unassigned variable.
- The goal test is to have all variables assigned while violating none of the constraints.
- Constraints should be represented explicitly, like $D \neq E$ for example.
- Use a function to test for constraint satisfaction.

15.5.1 8-queens example

- The row the first and second queen occupies: $V_1, V_2 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.
- No attack constraint for V_1 and V_2:

$$\{< 1, 3 >, < 1, 4 >, < 1, 5 >, \ldots, < 2, 4 >, < 2, 5 >, \ldots\}$$

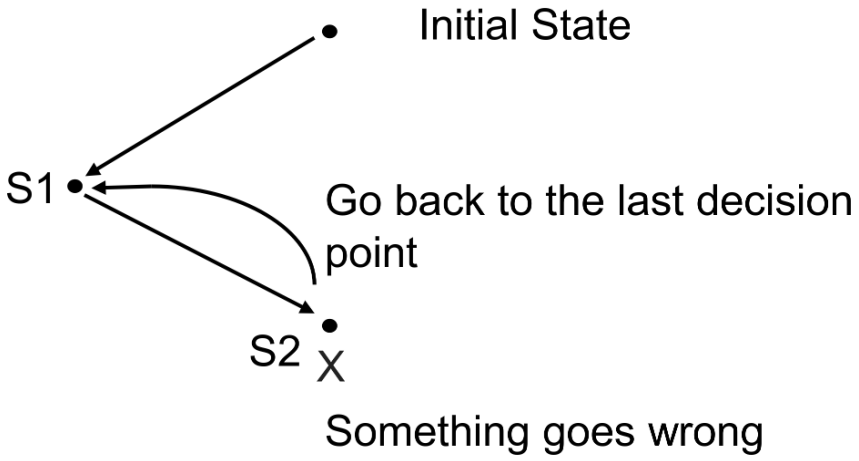
15.5.2 Map colouring example



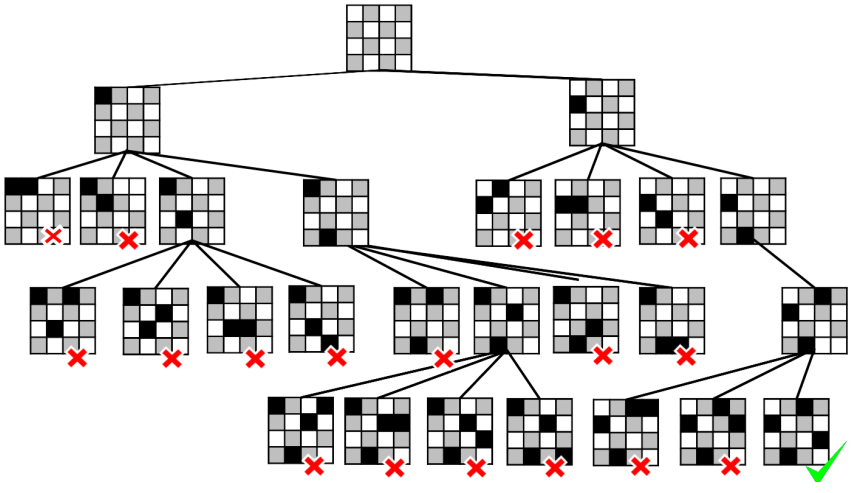
- Number of variables: n
- Maximum depth of the space: n
- Depth of solution state: n (all variables assigned)
- Search algorithm: Depth-first search

15.6 Backtracking search

- Backtracking search stops the search when constraints have already been violated.
- Before generating successors, check for constraint violations.
- If there are constraint violations, backtrack to try something else.



15.6.1 4-queens example

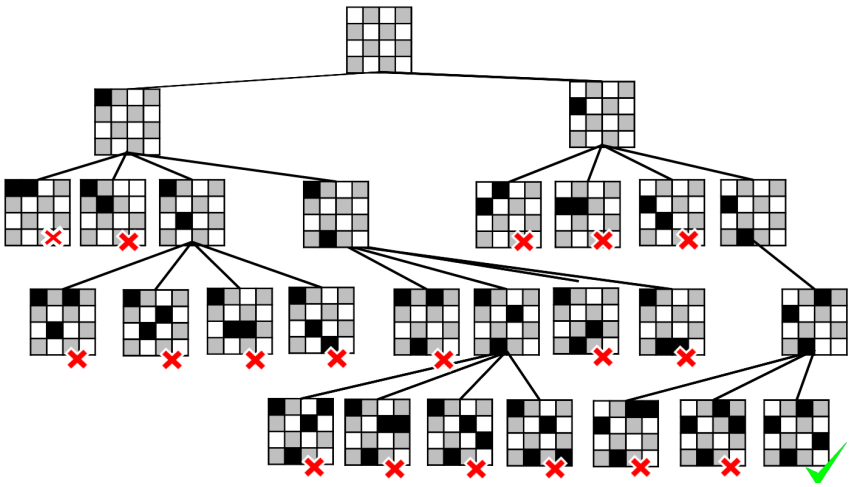


15.7 Heuristics for constraint satisfaction problems

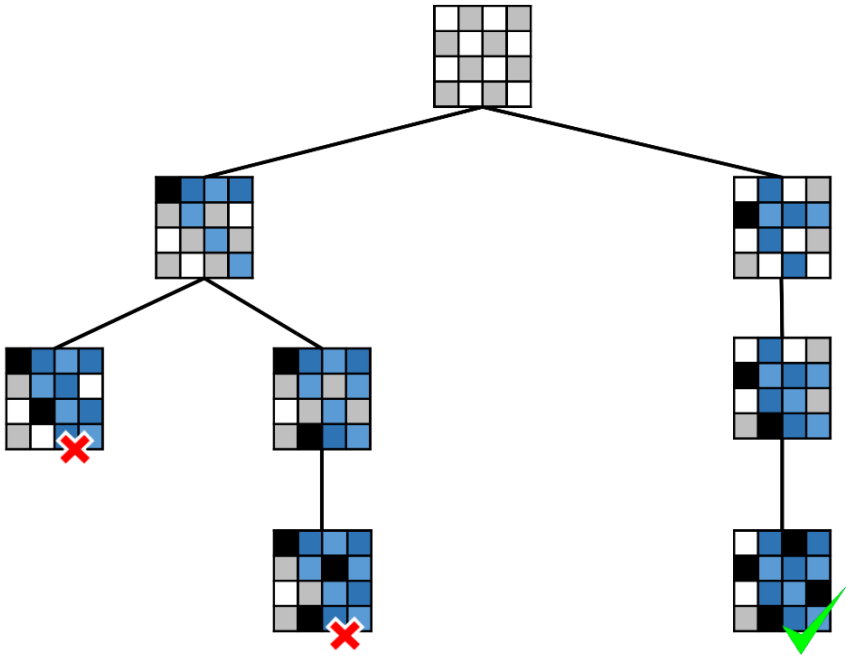
Plain backtracking is an uninformed algorithm. More intelligent kinds of search take into consideration:

- Which variable to assign next.
- What order of the values to try for each variable.
- Implications of current variable assignments for the other unassigned variables through forward checking and **constraint propagation**.
- **Constraint propagation** is propagating the implications of a constraint on one variable onto other variables.

15.7.1 4-queens example without constraint propagation

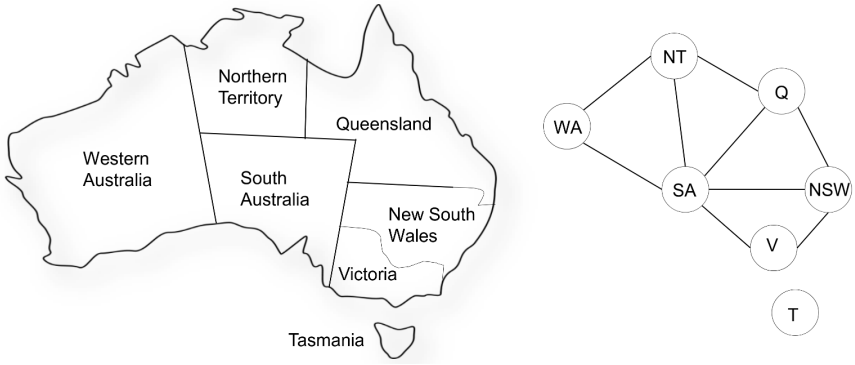


15.7.2 4-queens example with constraint propagation

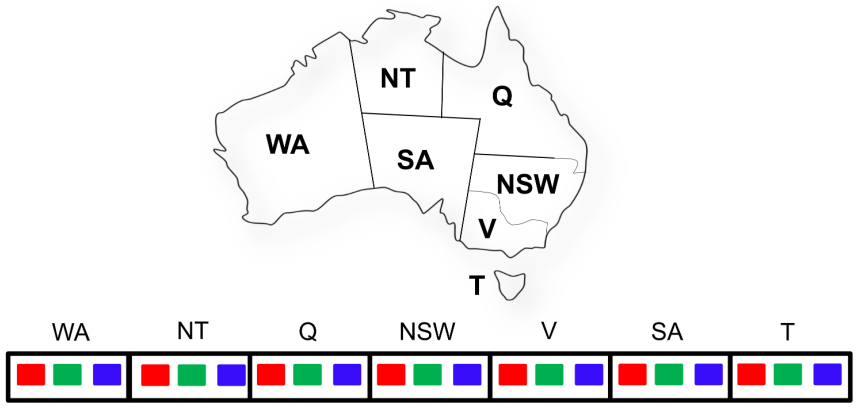


15.7.3 Map colouring example

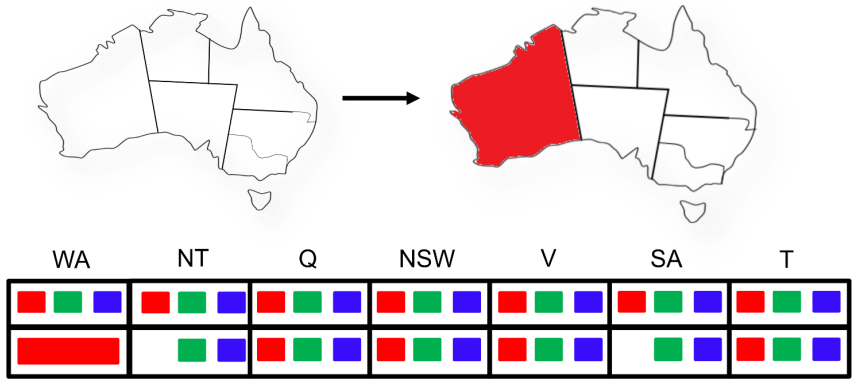
Graph representation:



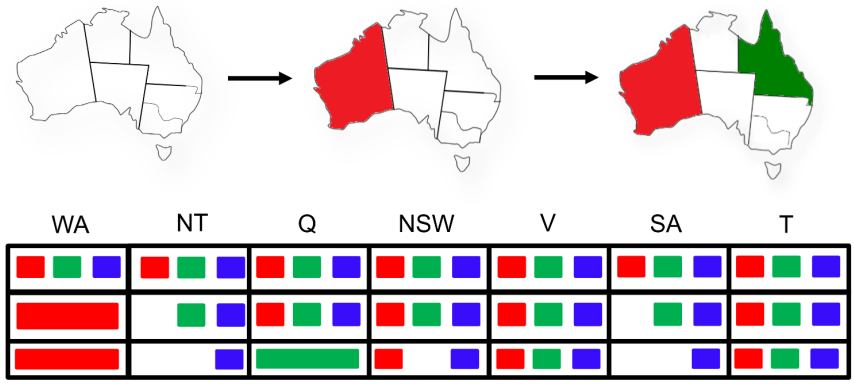
Initial state:



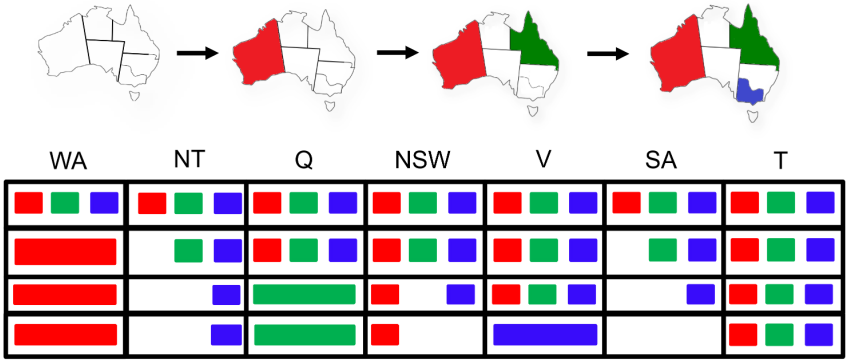
Step 1:



Step 2:



Step 3:

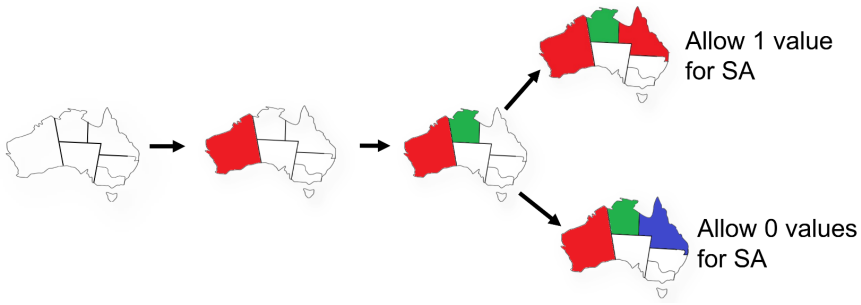


15.8 Most constrained variable

The most constrained variable, or the minimum remaining values (MRV) heuristic, is to reduce the branching factor on future choices by selecting the variable that is involved in the **largest number of constraints** on unassigned variables.

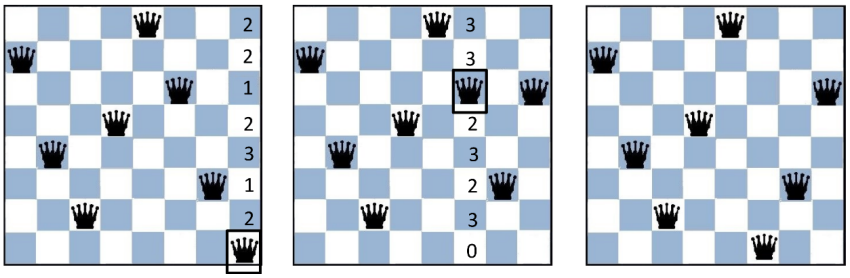
15.9 Least constraining value

The least constraining value heuristic is to choose the value that leaves maximum flexibility for subsequent variable assignments.



15.10 Minimum-conflicts heuristic (8-queens)

- The minimum-conflicts heuristic is a local heuristic search method for solving constraint satisfaction problems.
- Given an initial assignment, select a variable in the scope of a violated constraint and assign it to the value that minimises the number of violated constraints.



16 Games playing

16.1 Abstraction

- An abstraction is an ideal representation of real-world problems.
- Some examples include board games, chess, go, etc., as an abstraction of war games.
- Abstractions have perfect information, as they are fully observable.
- An abstraction that is accurately formulated represents the full state space of a problem.

16.2 Uncertainty

- Uncertainty takes into account the existence of hostile agents (players).
- These agents are acting to diminish the agent's well-being.
- Uncertainty about the actions of other agents is not due to the effect of non-deterministic actions and not due to randomness.
- This uncertainty is a contingency problem.

16.3 Complexity

- Games are abstract but not simple.
- For example, in chess, the average branching factor is 35, and chess has a game length greater than 50 moves, which makes its complexity 35^{50}, or 10^{40} for legal moves.
- Games are usually time-limited, which means a complete search for the optimal solution is not possible.
- There is hence uncertainty on the action's desirability, and search efficiency is crucial.

16.4 Types of games

Perfect information below refers to each player having complete information about the opponent's position, as well as the choices available to the opponent.

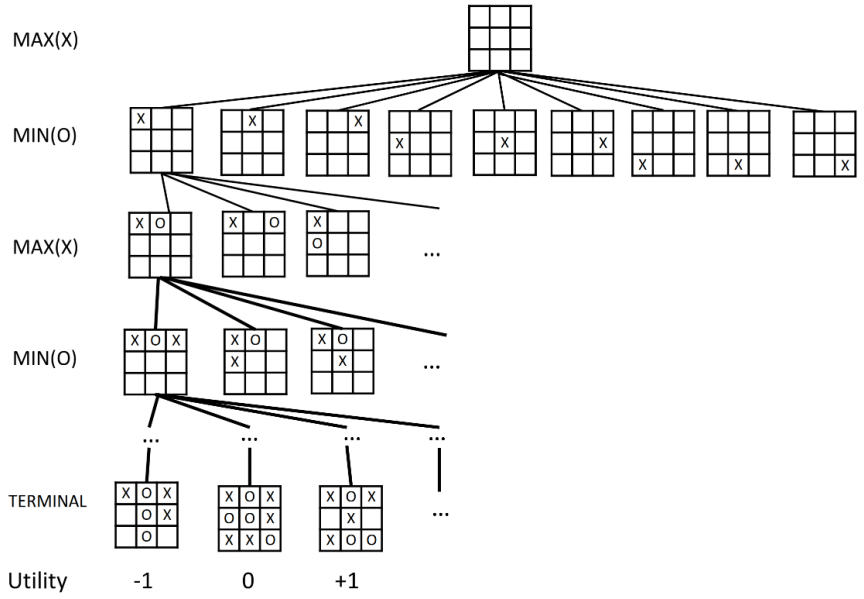
Information type	Deterministic	Chance
Perfect information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect information		Bridge, Poker, Scrabble, Nuclear war

16.5 Games as a search problem

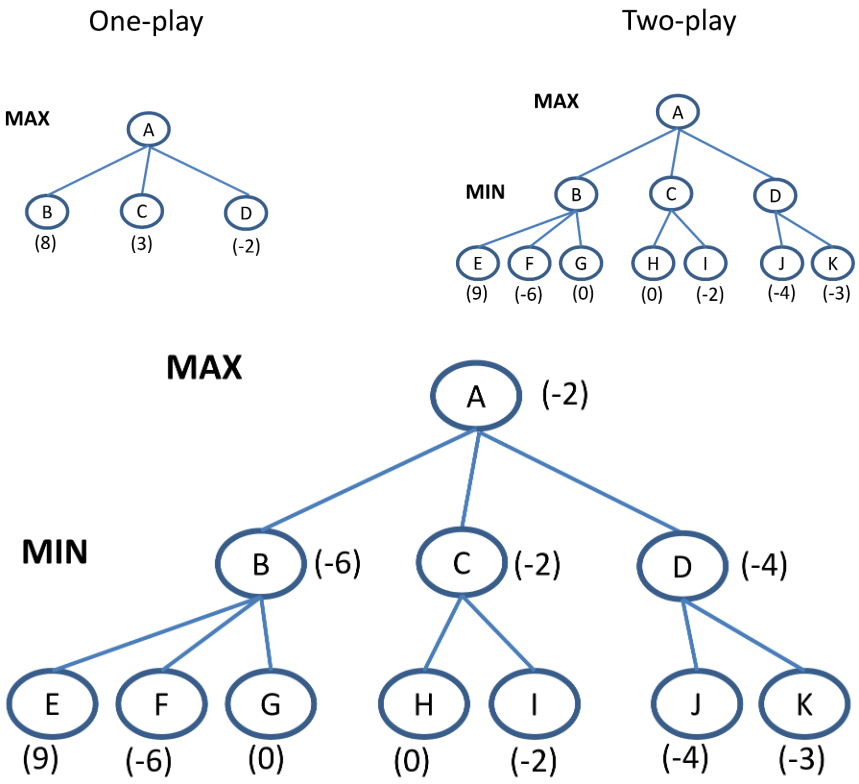
- The initial state is the initial board configuration and an indication of who makes the first move.
- The operators are the legal moves.
- The terminal test determines when the game is over.
- The utility function or the payoff function is a function that returns a numeric score to quantify the outcome of a game.
  - For example, in chess, +1 for a win, -1 for a loss, and 0 for a draw.

16.6 Game tree for tic-tac-toe

- The utility value of the terminal state is from the point of view of MAX.
- MAX uses the search tree to determine the best move.



16.7 Search strategy example



16.8 Minimax search strategy

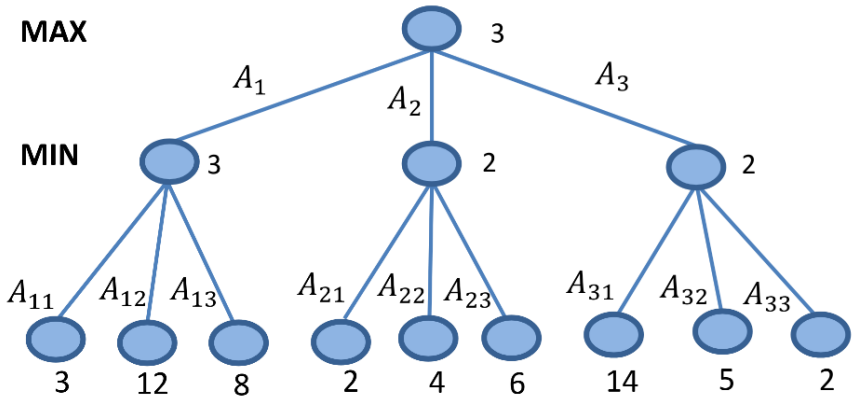
- A search strategy is defined as finding a sequence of moves that leads to a terminal state or a goal being achieved.
- Minimax search strategy maximises one's own utility and minimises the opponent's.
- The assumption of the minimax search strategy is that the opponent does the same.

16.8.1 Process

1. Generate the entire game tree down to the terminal states.
2. Calculate utility.
   (a) Assess the utility of each terminal state.
   (b) Determine the best utility of the parents of the terminal state.
   (c) Repeat the process for their parents until the root is reached.
3. Select the best move, which is the move with the highest utility value.

16.8.2 Perfect decisions

- A perfect decision is when no time limit is imposed, so the search algorithm can generate the complete search tree.
- There are two players, MAX and MIN.
  - Both players will choose moves with the best achievable payoff against the best play by the other player.
  - MAX tries to maximise the utility, assuming that MIN will try to minimise it.

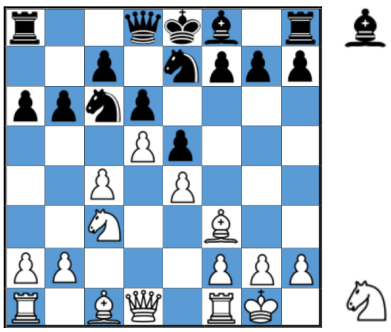


16.8.3 Imperfect decisions

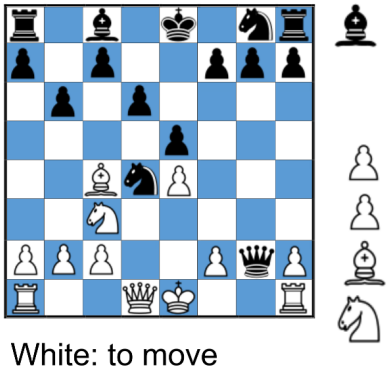
- For games like chess with a high branching factor, such as 35, each player typically makes 50 moves.
- In this case, the complete game tree would need to examine 35^{100} positions.
- With time and space requirements, a complete game tree search is intractable and hence it is impractical to make perfect decisions.
- With the above constraints, the minimax algorithm needs to be modified in two ways:
    1. The utility function needs to be replaced by an **estimated** desirability of the position, called an evaluation function.
    2. The full game tree search is replaced by a partial tree search with a depth limit.
    3. The terminal test used in the full game tree search is replaced by a cut-off test.

16.8.4 Evaluation functions

Evaluation functions return an estimate of the expected utility of the game from a given position.

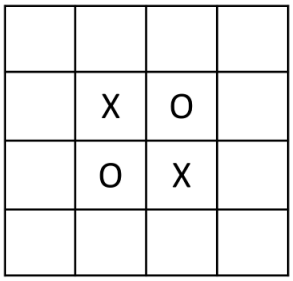


Black: to move
White: slightly better



White: to move
Black: winning

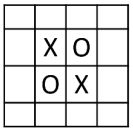
16.9 Othello 4



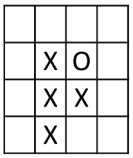
- A player can place a new piece in a position if there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another piece of the same kind, with one or more contiguous pieces from the opponent player between them.
- After placing the new piece, the pieces from the opponent player will be captured and become the pieces from the same player.
- The player with the most pieces on the board wins.

16.9.1 Game flow where "X" plays first

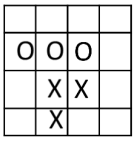
X considers the game now

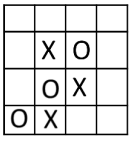


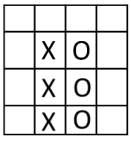
O considers the game now



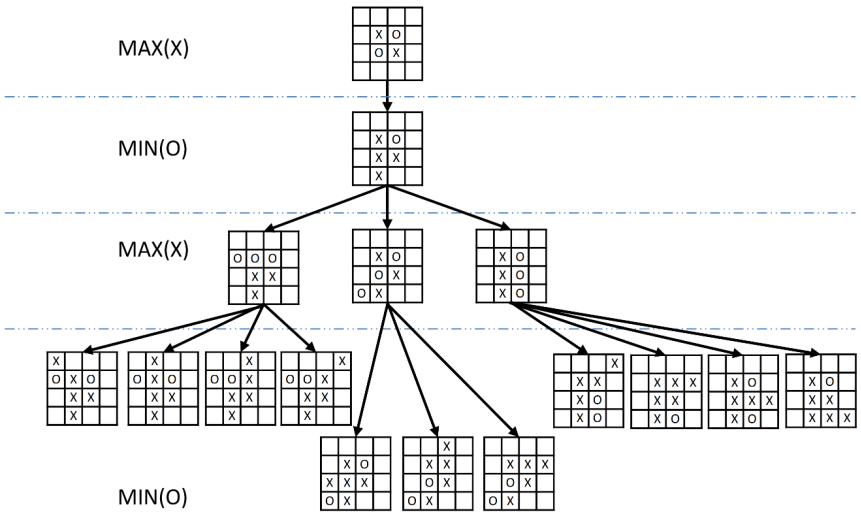
X considers the game now







16.9.2 Game tree



17 Agent decision-making and reinforcement learning

17.1 Maximum expected utility

The expected utility is a function of the utility function and the outcome probabilities, defined as follows:

$$EU(A|E) = \sum_i P(\text{Result}_i(A)|E,\ \text{Do}(A))U(\text{Result}_i(A))$$

The principle of maximum expected utility states to choose an action with the highest expected utility ($EU(A|E)$).

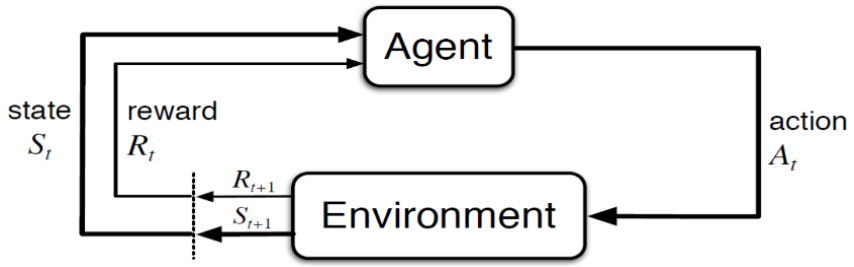
17.1.1 Example

For a robot with two options:

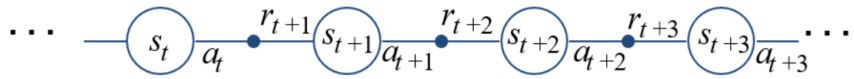
1. Turn right
    - Hits wall ($P = 0.1,\ U = 0$)
    - Finds target ($P = 0.9,\ U = 10$)
2. Turn left
    - Fall in water ($P = 0.3,\ U = 0$)
    - Finds target ($P = 0.7,\ U = 10$)

The robot should choose to turn right.

17.2 Agent-environment interface



- The agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$
- It observes state at step t : $s_t \in S$
- It produces action at step t : $a_t \in A(s_t)$
- It gets a resulting reward: $r_{t+1} \in \mathbb{R}$
- It also gets the resulting next state: s_{t+1}



17.3 Making complex decisions

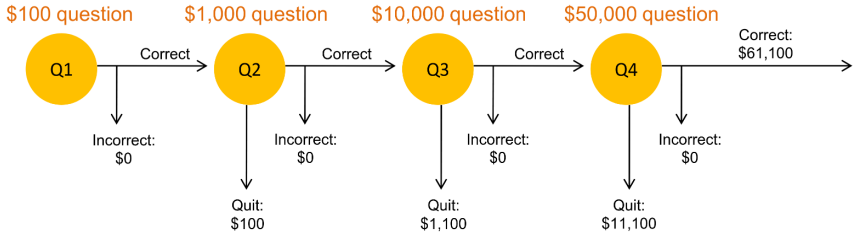
- Make a sequence of decisions.
  - Agent's utility depends on a sequence of decisions.
  - Sequential decision-making.
- Markov property
  - Transition property depends only on the current state, not on previous history (how that state was reached).
  - Markov decision processes.

17.3.1 Markov decision processes

- Components:
  - Markov **states** s, beginning with the initial state s_0.
  - **Actions** a, where each state has actions $A(s)$ available from it.
  - **Transition model** $P(s'|s,a)$, where the assumption is that the probability of going from s' from s depends only on s and a and not on any other past actions or states.
  - **Reward function** $R(s)$.
- **Policy** $\pi(s)$, which is the action that an agent takes in any given state. This is the "solution" to a Markov decision process.

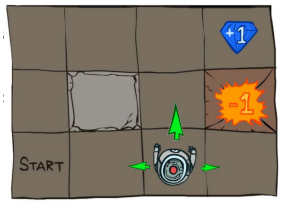
17.4 Game show

- A series of questions with increasing levels of difficulty and payoff.
- At each step, take your earnings and quit, or go for the next question. If you answer wrong, you lose everything.



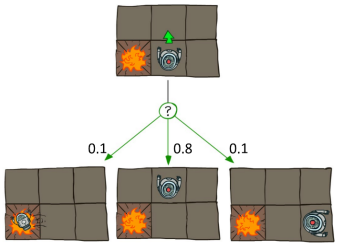
- The probability of guessing correctly is 0.1.
- Expected payoff for continuing is $0.1 \times 61,100 + 0.9 \times 0 = 6,110$.
- The optimal decision would be to quit.
- In question 3:
  - Payoff for quitting: \$1,100
  - Payoff for continuing: $0.5 \times 11,100 = 5,550$
- In question 2:
  - \$100 for quitting versus \$4,162 for continuing.
- In question 1:
  - \$0 for quitting versus \$100 for continuing.

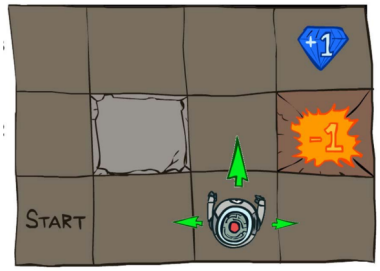
17.5 Grid world

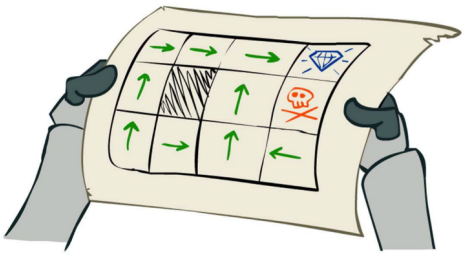


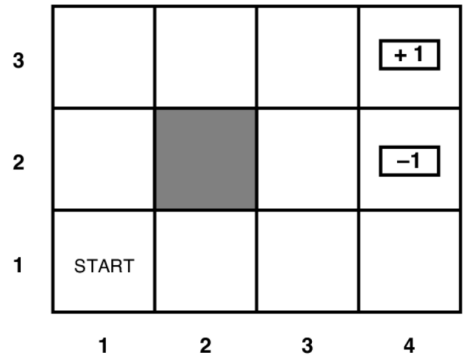
R(s) = -0.04 for every non-terminal state

Transition model:

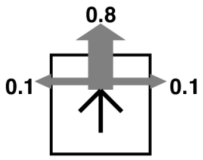




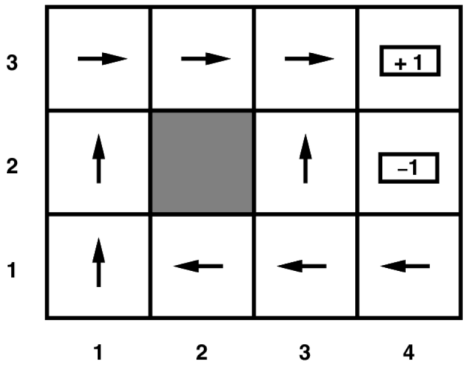




Transition model:



R(s) = -0.04 for every non-terminal state



Optimal policy when R(s) = -0.04 for every non-terminal state

17.6 Solving Markov decision processes (MDPs)

- MDP components:
  - **States** s
  - **Actions** a
  - **Transition model** $P(s'|s,a)$
  - **Reward function** $R(s)$
- The solution:
  - **Policy** $\pi(s)$: mapping from states to actions

17.6.1 Maximising expected utility

- The optimal policy should maximise the expected utility over all possible state sequences produced by following that policy:

$$\sum_{\text{State sequences starting from } s_0} P(\text{sequence})U(\text{sequence})$$

- The utility of a state sequence is the sum of rewards of individual states.
- However, there is an issue with infinite state sequences.

17.6.2 Utility of state sequences

- Usually, the utility of a state sequence is the sum of the rewards of the individual state.
- However, due to the infinite state sequences problem, each of the rewards of the individual states has to be discounted by a factor γ between 0 and 1.

$$\begin{aligned} U([s_0, s_1, s_2, \ldots]) &= R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots \\ &= \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \frac{R_{max}}{1-\gamma} \quad (0 < \gamma < 1) \end{aligned}$$

  - Sooner rewards count more than later rewards.
  - This makes sure the total utility stays bounded.
  - It helps algorithms converge.

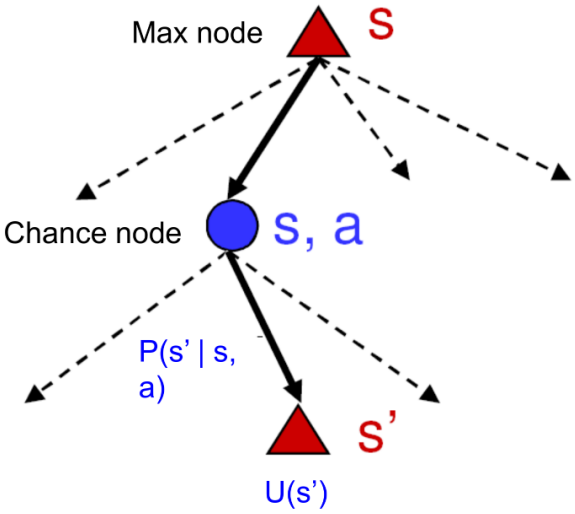
17.6.3 Utility of states

- Expected utility obtained by policy π starting in state s is:

$$U^{\pi}(s) = \sum_{\text{State sequences starting from } s} P(\text{sequence})U(\text{sequence})$$

- The "true" utility of a state, denoted $U(s)$, is the expected sum of discounted rewards if the agent executes an optimal policy starting in state s.
- This is reminiscent of the minimax values of states.

17.6.4 Finding the utilities of states



- The expected utility of taking action a in state s is:

$$\sum_{s'} P(s'|s,a)U(s')$$

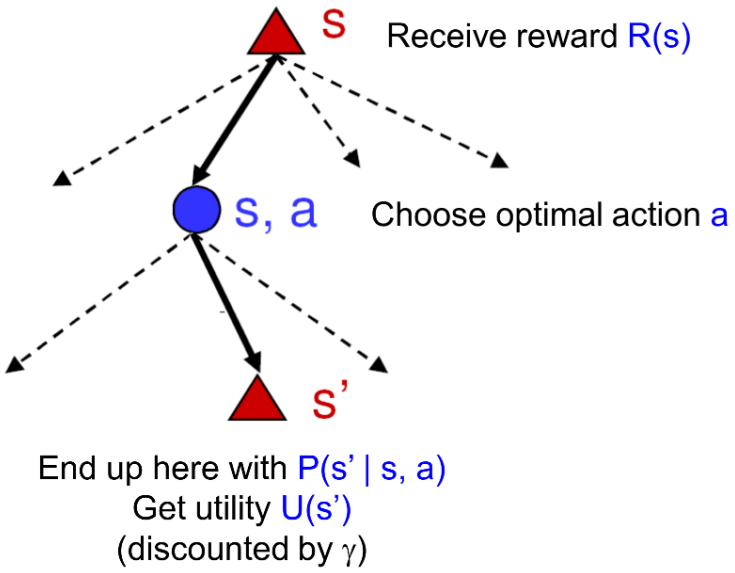
- The optimal action is given by:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

- The recursive expression for $U(s)$ in terms of the utilities of its successor states is:

$$U(s) = R(s) + \gamma \max_{a} \sum_{s'} P(s'|s,a)U(s')$$

17.7 Bellman equation



- The recursive relationship between the utilities of successive states:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

- For N states, we get N equations in N unknowns.
  - Solving them solves the Markov decision process.
  - Trying to solve them using expectimax search would run into issues with infinite sequences.
  - Instead, they should be solved algebraically.
  - There are two methods to doing so, **value iteration** and **policy iteration**.

17.7.1 Value iteration

- Start out with every $U(s) = 0$.
- Iterate until convergence.
  - During the i-th iteration, update the utility of each state according to the rule below:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

- In the limit of infinitely many iterations, the correct utility values are guaranteed to be found.
- In practice, however, there is no need to use an infinite number of iterations.

17.7.2 Policy iteration

- Start with some initial policy π_0 and alternate between the following steps:
  - **Policy evaluation**: Calculate $U^{\pi_i}(s)$ for every state s.
  - **Policy improvement**: Calculate a new policy π_{i+1} based on the updated utilities.

$$\pi^{i+1}(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a) U^{\pi_i}(s')$$

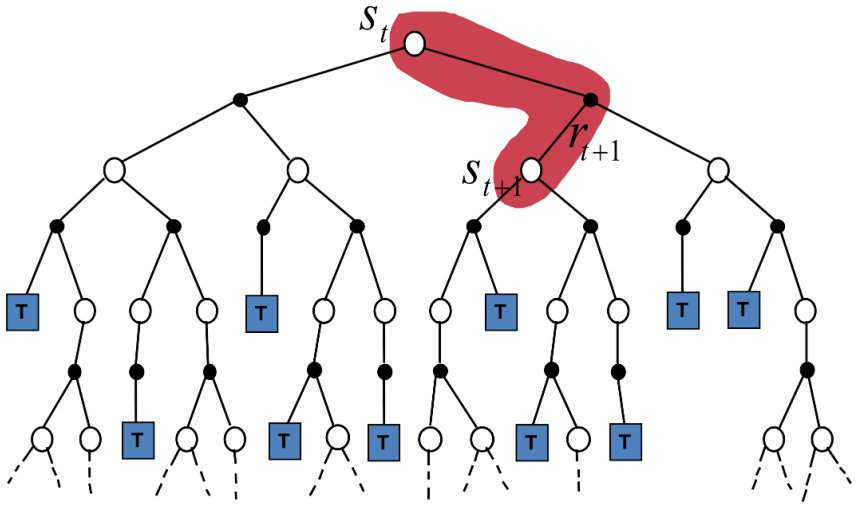
17.8 Temporal difference prediction

- Policy evaluation (the prediction problem):
  - For a given policy p, compute the state-value function V^π.
  - The simplest temporal difference method, $TD(0)$:

$$V(s_t) \leftarrow v(s_t) + \alpha[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{Target: an estimate of the return}} - V(s_t)]$$

17.8.1 Simplest temporal difference method

$$V(s_t) \leftarrow v(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$



17.8.2 Advantages of temporal difference learning

- Temporal difference methods do not require a model of the environment, only experience.
- Temporal difference methods can be fully incremental.
  - The model can learn **before** knowing the final outcome, which results in less memory used and lower peak computation.
  - The model can learn **without** the final outcome, which means it can learn from incomplete sequences.

18 Python reference

18.1 Default imports

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

# Set the theme for seaborn
sb.set_theme()
```

18.2 Loading the data

```
data = pd.read_csv("./data.csv")
print(data.shape)
```

```
(1460, 81)
```

18.3 Selecting all variables of a type

```
numeric_data = data.select_dtypes("number")
print(numeric_data.shape)
```

```
(1460, 38)
```

Get the actual numeric data from the data description.

```
numeric_variables = [
    "LotArea",
    "GrLivArea",
    "TotalBsmtSF",
    "GarageArea",
    "SalePrice"
]
numeric_data = data[numeric_variables]
print(numeric_data.head())
```

```
   LotArea  GrLivArea  TotalBsmtSF  GarageArea  SalePrice
0     8450       1710          856         548     208500
1     9600       1262         1262         460     181500
2    11250       1786          920         608     223500
3     9550       1717          756         642     140000
4    14260       2198         1145         836     250000
```

18.4 Head of the data

```
print(numeric_data.head())
```

```
   LotArea  GrLivArea  TotalBsmtSF  GarageArea  SalePrice
0     8450       1710          856         548     208500
1     9600       1262         1262         460     181500
2    11250       1786          920         608     223500
3     9550       1717          756         642     140000
4    14260       2198         1145         836     250000
```

18.5 Statistical description of the data

```
print(numeric_data.describe())
```

```
             LotArea    GrLivArea  TotalBsmtSF   GarageArea      SalePrice
count    1460.000000  1460.000000  1460.000000  1460.000000    1460.000000
mean    10516.828082  1515.463699  1057.429452   472.980137  180921.195890
std      9981.264932   525.480383   438.705324   213.804841   79442.502883
min      1300.000000   334.000000     0.000000     0.000000   34900.000000
25%      7553.500000  1129.500000   795.750000   334.500000  129975.000000
50%      9478.500000  1464.000000   991.500000   480.000000  163000.000000
75%     11601.500000  1776.750000  1298.250000   576.000000  214000.000000
max    215245.000000  5642.000000  6110.000000  1418.000000  755000.000000
```

18.6 Skew of the data

```
print(numeric_data.skew())
```

```
LotArea        12.207688
GrLivArea       1.366560
TotalBsmtSF     1.524255
GarageArea      0.179981
SalePrice       1.882876
dtype: float64
```

18.7 Getting the number of outliers

```
quartile_1 = numeric_data.quantile(0.25)
quartile_3 = numeric_data.quantile(0.75)
interquartile_range = quartile_3 - quartile_1

number_of_outliers = (
    (numeric_data < quartile_1 - 1.5 * interquartile_range)
    | (numeric_data > quartile_3 + 1.5 * interquartile_range)
).sum()

print("Number of outliers:")
print(number_of_outliers)
```

```
Number of outliers:
LotArea        69
GrLivArea      31
TotalBsmtSF    61
GarageArea     21
SalePrice      61
dtype: int64
```

18.8 Correlation matrix

```
print(numeric_data.corr())
```

```
              LotArea  GrLivArea  TotalBsmtSF  GarageArea  SalePrice
LotArea      1.000000   0.263116     0.260833    0.180403   0.263843
GrLivArea    0.263116   1.000000     0.454868    0.468997   0.708624
TotalBsmtSF  0.260833   0.454868     1.000000    0.486665   0.613581
GarageArea   0.180403   0.468997     0.486665    1.000000   0.623431
SalePrice    0.263843   0.708624     0.613581    0.623431   1.000000
```

18.9 Converting data types

```
categorical_variables = [
    "MSSubClass",
    "Neighborhood",
    "BldgType",
    "OverallQual",
]

categorical_data = data[categorical_variables]
print("Data types:")
print(categorical_data.dtypes)
print()

categorical_data = categorical_data.astype("category")
print("Data types:")
print(categorical_data.dtypes)
```

```
Data types:
MSSubClass       int64
Neighborhood    object
BldgType        object
OverallQual      int64
dtype: object

Data types:
MSSubClass      category
Neighborhood    category
BldgType        category
OverallQual     category
dtype: object
```

18.10 Grouping data

```
# Need to call ".size" on the grouped data
# to actually group the data.
grouped_data = data.groupby(
    by=["OverallQual", "BldgType"],

    # Get rid of the deprecation warning
    observed=False,
).size()

# Unstack the data so there is a 2D array
# that is plottable.
grouped_data = grouped_data.unstack()

print(grouped_data.head())
```

```
BldgType      1Fam  2fmCon  Duplex  Twnhs  TwnhsE
OverallQual
1              2.0     NaN     NaN    NaN     NaN
2              3.0     NaN     NaN    NaN     NaN
3             16.0     1.0     3.0    NaN     NaN
4             90.0     8.0     8.0    6.0     4.0
5            339.0    15.0    31.0    3.0     9.0

BldgType      1Fam  2fmCon  Duplex  Twnhs  TwnhsE
OverallQual
1              2.0     NaN     NaN    NaN     NaN
2              3.0     NaN     NaN    NaN     NaN
3             16.0     1.0     3.0    NaN     NaN
4             90.0     8.0     8.0    6.0     4.0
5            339.0    15.0    31.0    3.0     9.0
```

18.11 Splitting data randomly

The code below imports the `train_test_split` function and splits the data set into 70% training data and 30% test data.

```
from sklearn.model_selection import train_test_split
training_data, test_data = train_test_split(data, test_size=0.3)
print(training_data.shape)
print(test_data.shape)
```

```
(1022, 81)
(438, 81)
```

18.12 Linear regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

18.12.1 Training the model

```
training_data_x = pd.DataFrame(training_data["GrLivArea"])
training_data_y = pd.DataFrame(training_data["SalePrice"])
model.fit(training_data_x, training_data_y)
```

18.12.2 Scoring the model

```
accuracy = model.score(training_data_x, training_data_y)
print(accuracy)
```

```
0.518488052675584
```

18.12.3 Predicting data

```
prediction = model.predict(training_data_x)
print(prediction)
```

```
[[112899.37012923]
 [177936.06711062]
 [246798.45214974]
 ...
 [205391.00493668]
 [232170.82134078]
 [227444.97138711]]
```

18.12.4 Getting the model coefficients after training

```
model_coefficients = model.coef_[0][0]
print(model_coefficients)
```

```
112.5202369920301
```

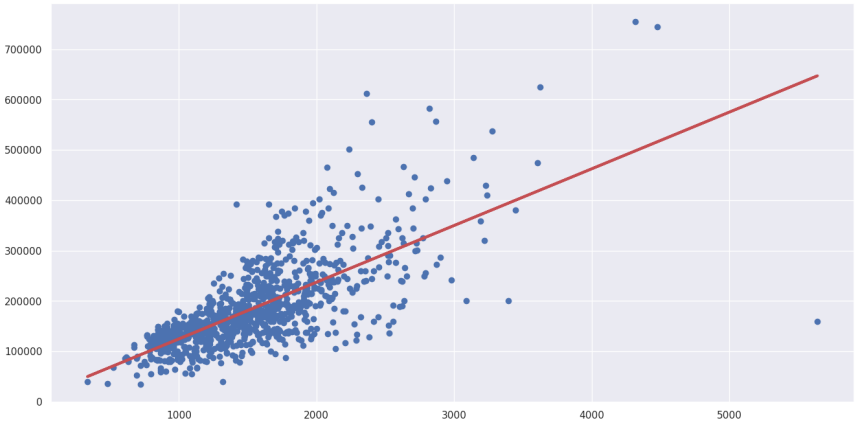
18.12.5 Forming the equation of the line

```
regression_x = training_data_x
regression_y = model_coefficients * regression_x + model.intercept_
print(regression_x.head())
print()
print(regression_y.head())
```

```
      GrLivArea
1215        894
1018       1472
787        2084
309        1944
988        2030

          GrLivArea
1215  112899.370129
1018  177936.067111
787   246798.452150
309   231045.618971
988   240722.359352
```

18.12.6 Plotting the graph

```
plt.figure(figsize=(16, 8))
plt.scatter(training_data_x, training_data_y)
plt.plot(regression_x, regression_y, "r-", linewidth=3)
```



18.13 K-means clustering

The number of clusters should be guessed based on the data.

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=4)
```

18.13.1 Training the model

```
extracted_data = data[["GrLivArea", "GarageArea"]]
model.fit(extracted_data)
```

18.13.2 Getting the cluster centres

```
print(pd.DataFrame(model.cluster_centers_))
```

```
             0           1
0  1557.328479  490.318770
1  2147.264925  625.462687
2  3178.974359  707.717949
3  1029.347664  359.456075
```

18.13.3 Predicting the clusters for the data

```
predicted_clusters = model.predict(extracted_data)
print(pd.DataFrame(predicted_clusters).head())
```

```
   0
0  0
1  3
2  0
3  0
4  1
```

18.13.4 Adding the predicted clusters to the data

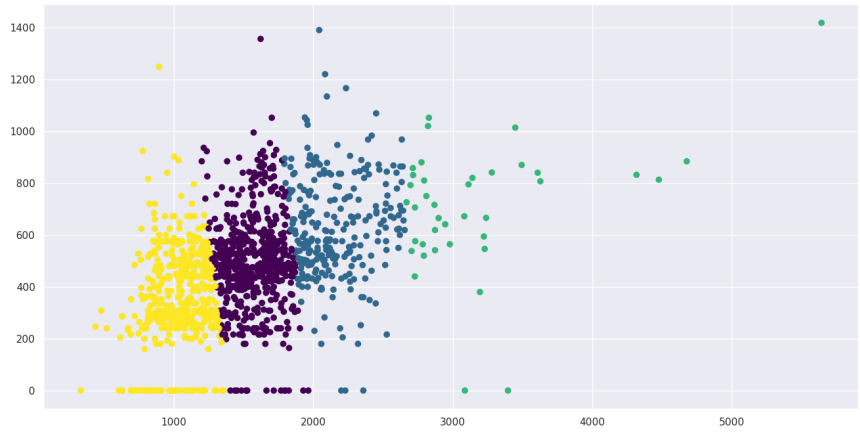
```
clustered_data = extracted_data.copy()
clustered_data["Cluster"] = pd.Categorical(predicted_clusters)
print(clustered_data.head())
```

```
   GrLivArea  GarageArea Cluster
0       1710         548       0
1       1262         460       3
2       1786         608       0
3       1717         642       0
4       2198         836       1
```

18.13.5 Visualising the clusters in a scatter plot

You can view all the possible colour palettes here.

```
plt.subplots(figsize=(16, 8))
plt.scatter(
    x="GrLivArea",
    y="GarageArea",

    # Colour the points
    # based on the cluster
    c="Cluster",

    # Set a colour map
    cmap="viridis",
    data=clustered_data,
)
```



18.13.6 Getting the ideal number of clusters

First, we create a function to calculate the square of the Euclidean distance of one point away from another point.

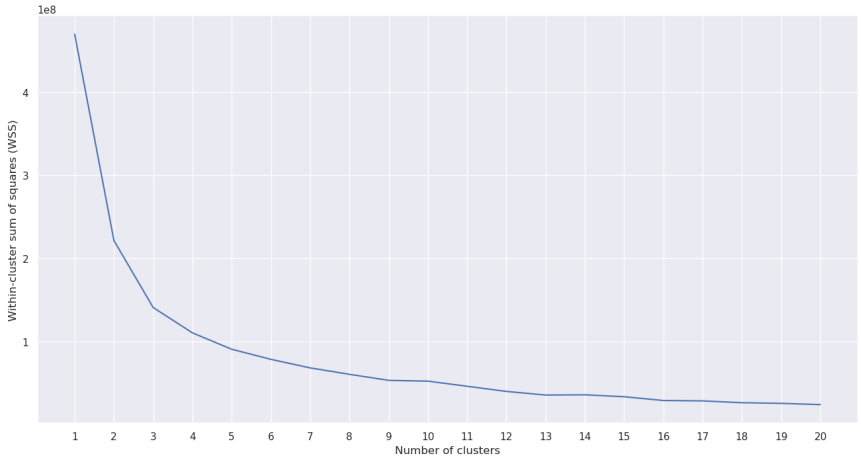
```
def get_squared_euclidean_distance(
    point_1: np.ndarray,
    point_2: np.ndarray,
) -> float:
    """
    A function to get the square of the Euclidean distance between two points.

    The first point and the second point must have the
    same dimensions, i.e. if point_1 is 2 dimensional, with coordinates (x_1, y_1),
    then point_2 should also be 2 dimensional, with coordinates (x_2, y_2).
    """

    # Initialise the squared Euclidean distance
    squared_euclidean_distance = 0

    # Iterate over the coordinates
    for index, point_1_coordinate in enumerate(point_1):

        # Get the coordinate for the second point
        point_2_coordinate = point_2[index]

        # Subtract the coordinates of point 1
        # from the coordinates of point 2,
        # and square the result
        coordinate_euclidean_distance = (
            point_2_coordinate - point_1_coordinate
        ) ** 2

        # Add the Euclidean distance for the coordinate
        # to the squared Euclidean distance
        squared_euclidean_distance += coordinate_euclidean_distance

    # Return the squared Euclidean distance
    return squared_euclidean_distance
```

Next, we iterate over the possible number of clusters and fit the k-Means algorithm, while calculating the within-cluster sum of squares.

```
# The possible number of clusters
possible_number_of_clusters = list(range(1, 21))

# Initialise the list of the within-cluster sum of squares (WSS)
within_cluster_sum_of_squares_list = []

# Iterate over the number of clusters
for number_of_clusters in possible_number_of_clusters:

    # Initialise the K means clustering model with
    # the number of clusters and fit the model to the data
    model = KMeans(n_clusters=number_of_clusters).fit(extracted_data)

    # Get the centres and predict the clusters for the data
    centres = model.cluster_centers_
    predicted_clusters = model.predict(extracted_data)

    # Initialise the variable to store the within-cluster sum of squares
    within_cluster_sum_of_squares = 0

    # Iterate over the points in the data
    for point_index, point in enumerate(np.array(extracted_data)):

        # Get the centre for the current point
        point_centre = centres[predicted_clusters[point_index]]

        # Calculate the squared Euclidean distance
        squared_euclidean_distance = get_squared_euclidean_distance(
            point_centre, point
        )

        # Add the squared Euclidean distance for the point
        # to the within-cluster sum of squares
        within_cluster_sum_of_squares += squared_euclidean_distance

    # Add the within-cluster sum of squares to the list
    within_cluster_sum_of_squares_list.append(within_cluster_sum_of_squares)
```

Lastly, we plot the graph.

```
# Plot the graph of within-cluster sum of squares (y)
# against the number of clusters (x)
plt.figure(figsize=(16, 8))
plt.plot(
    possible_number_of_clusters,
    within_cluster_sum_of_squares_list,
)

# Force the x-tick labels to display the number of clusters,
# which are integers, instead of the default
plt.xticks(possible_number_of_clusters)

# Label the x and y axis
plt.xlabel("Number of clusters")
plt.ylabel("Within-cluster sum of squares (WSS)")
```



Using the elbow method, we can see that the ideal number of clusters here is 3 clusters.

18.14 Local outlier factor

- The parameters for the local outlier factor model should be given.
- The `contamination` is just the fraction or percentage of outliers.

```
from sklearn.neighbors import LocalOutlierFactor
model = LocalOutlierFactor(n_neighbors=20, contamination=0.05)
```

18.14.1 Training the model

```
model.fit(extracted_data)
```

18.14.2 Predicting anomalies

```
labels = model.fit_predict(extracted_data)
print(pd.DataFrame(labels).head())
```

```
   0
0  1
1  1
2  1
3  1
4  1
```

18.14.3 Adding anomalies to the data

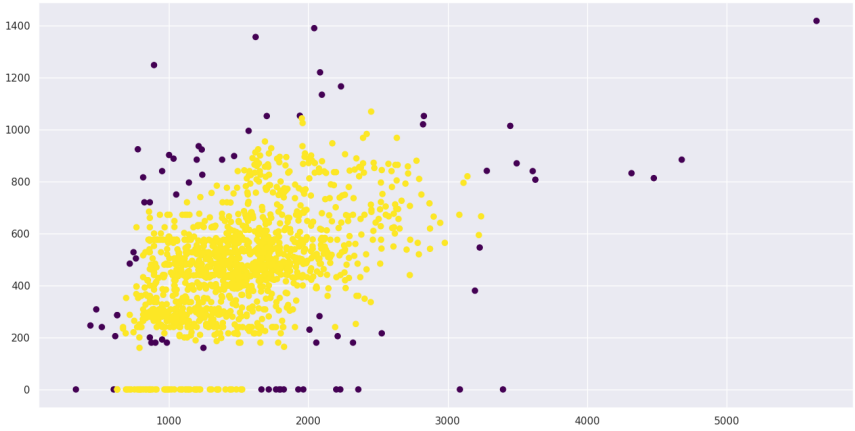
```
labelled_data = extracted_data.copy()
labelled_data["Anomaly"] = pd.Categorical(labels)
print(labelled_data.head())
```

```
   GrLivArea  GarageArea Anomaly
0       1710         548       1
1       1262         460       1
2       1786         608       1
3       1717         642       1
4       2198         836       1
```

18.14.4 Visualising the anomalies in the data

You can view all the possible colour palettes here.

```
plt.subplots(figsize=(16, 8))
plt.scatter(
    x="GrLivArea",
    y="GarageArea",

    # Colour the points
    # based on whether it is
    # an anomaly or not
    c="Anomaly",

    # Set a colour map
    cmap="viridis",
    data=labelled_data
)
```



18.15 Decision tree

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(max_depth=2)
```

18.15.1 Training the model

```
training_data_x = pd.DataFrame(training_data["SalePrice"])
training_data_y = pd.DataFrame(training_data["CentralAir"])
model.fit(training_data_x, training_data_y)
```

18.15.2 Scoring the model

```
accuracy = model.score(training_data_x, training_data_y)
print(accuracy)
```

```
0.9500978473581213
```

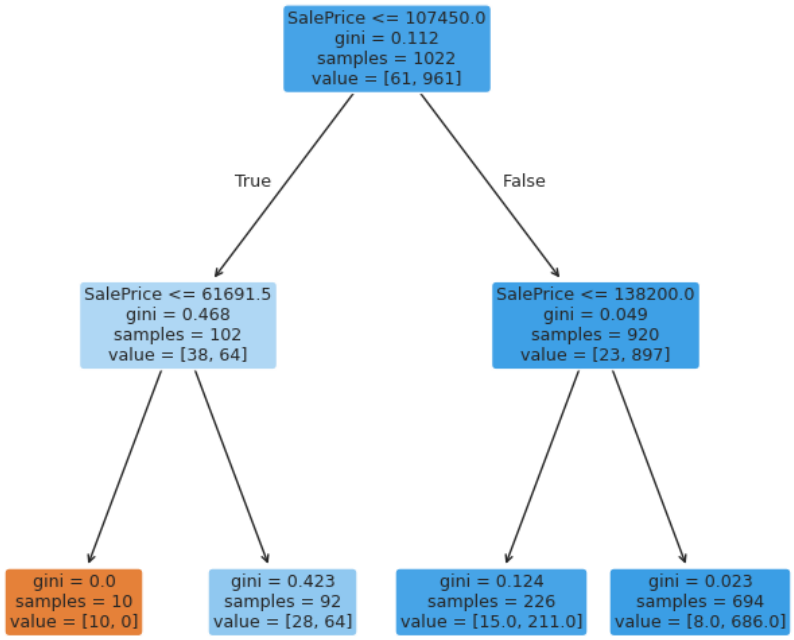
18.15.3 Predicting data

```
prediction = model.predict(training_data_x)
print(prediction)
```

```
['Y' 'Y' 'Y' ... 'Y' 'Y' 'Y']
```

18.15.4 Plotting the decision tree

```
from sklearn.tree import plot_tree
plt.figure(figsize=(8, 8))
plot_tree(
    model,
    filled=True,
    rounded=True,
    feature_names=["SalePrice"]
)
```



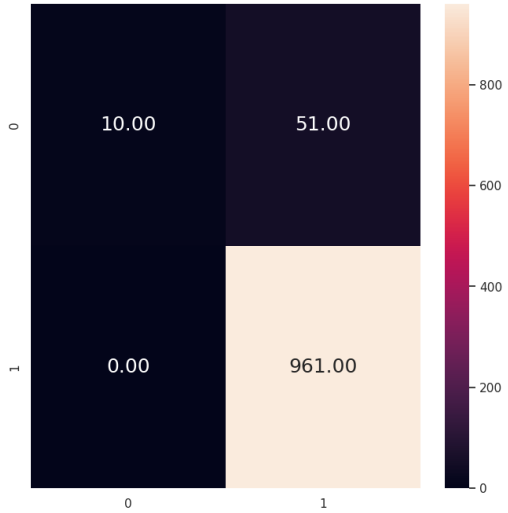
18.16 Confusion matrix

```
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(training_data_y, prediction)
print(matrix)
```

```
[[ 10  51]
 [  0 961]]
```

18.16.1 Plotting the confusion matrix

```
plt.figure(figsize=(8, 8))
sb.heatmap(
    matrix,

    # Annotate the squares
    # in the heatmap
    annot=True,

    # Round the values to 2 decimal places
    fmt=".2f",

    # Set the size of the
    # annotations to 18
    annot_kws={"size": 18},
)
```



18.16.2 Getting the true and false positives

```
tp, fp, fn, tn = matrix.ravel()
print(f"True positives: {tp}")
print(f"False positives: {fp}")
print(f"False negatives: {fn}")
print(f"True negatives: {tn}")
```

```
True positives: 10
False positives: 51
False negatives: 0
True negatives: 961
```

18.16.3 Getting the accuracy measures

```
tpr = tp / (tp + fn)
tnr = tn / (tn + fp)
fpr = fp / (fp + tn)
fnr = fn / (tp + fn)

print(f"True positive rate: {tpr}")
print(f"True negative rate: {tnr}")
print(f"False positive rate: {fpr}")
print(f"False negative rate: {fnr}")
```

```
True positive rate: 1.0
True negative rate: 0.9496047430830039
False positive rate: 0.05039525691699605
False negative rate: 0.0
```

18.16.4 Calculating the precision

```
precision = tp / (tp + fp)
print(precision)
```

```
0.16393442622950818
```

18.16.5 Calculating the recall

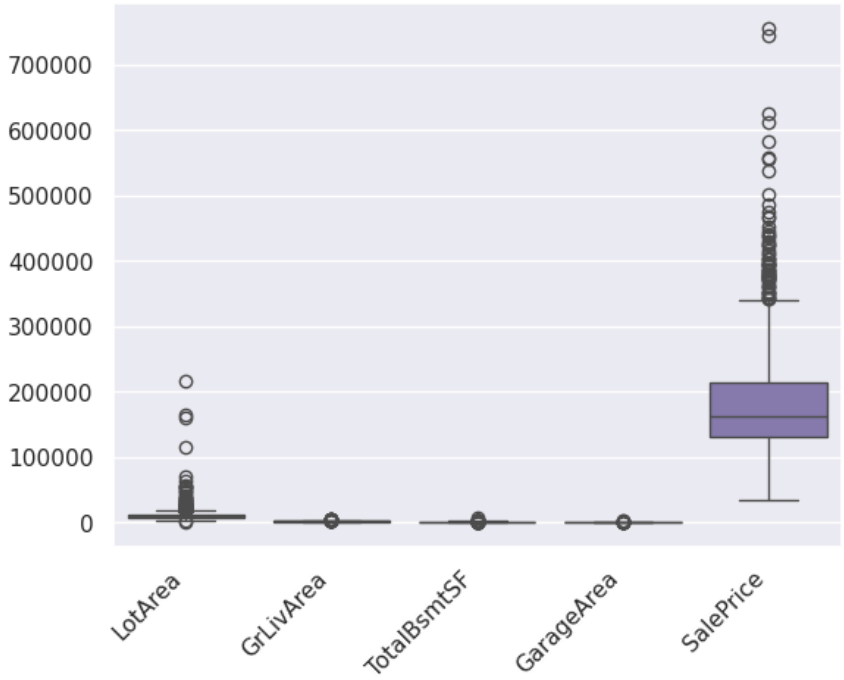
```
recall = tp / (tp + fn)
print(recall)
```

```
1.0
```

18.17 Graphs

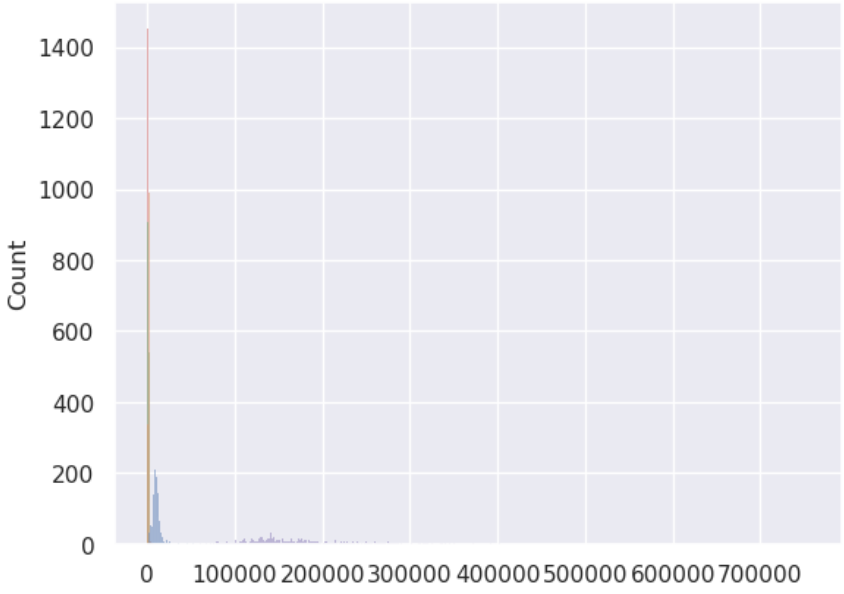
18.17.1 Dealing with long labels

```
# Plot any graph that displays vertically
sb.boxplot(numeric_data)

# Rotate the x-axis labels by 45 degrees to the right
plt.xticks(rotation=45, ha="right")
```



18.17.2 Removing the legend

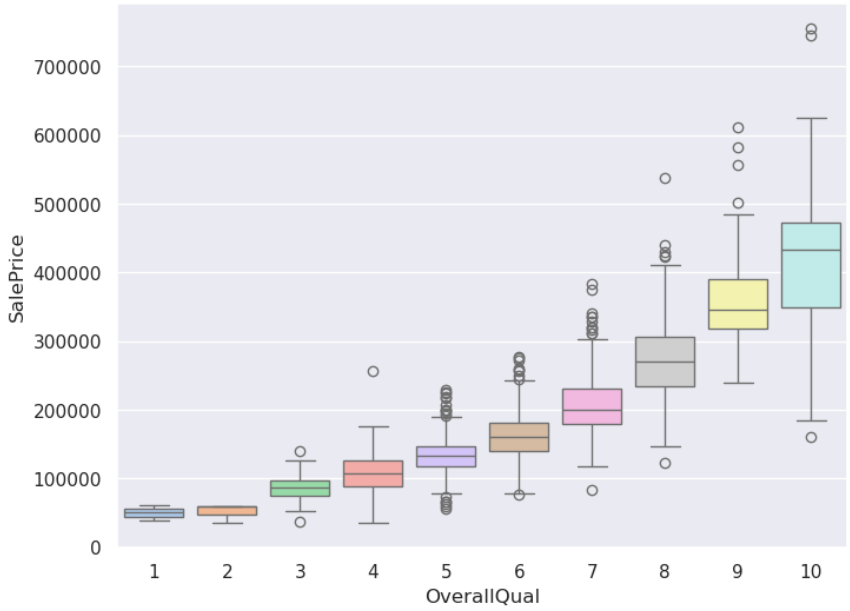
```
# Plot any graph
sb.histplot(numeric_data)

# Remove the legend
plt.legend([], [], frameon=False)
```



18.17.3 Box plot

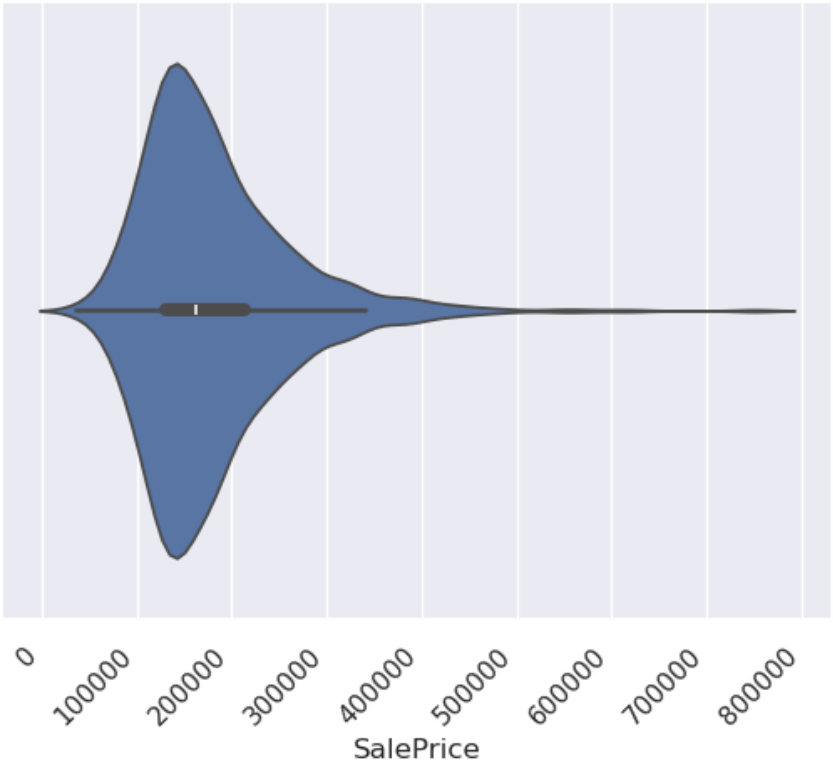
You can view all the possible colour palettes here.

```
plt.figure(figsize=(8, 6))
sb.boxplot(
    data,
    x="OverallQual",
    y="SalePrice",

    # Set the hue to a category to colour it
    hue="OverallQual",

    # Set a colour palette
    palette="pastel",
)

# Remove the legend
plt.legend([], [], frameon=False)
```



18.17.4 Violin plot

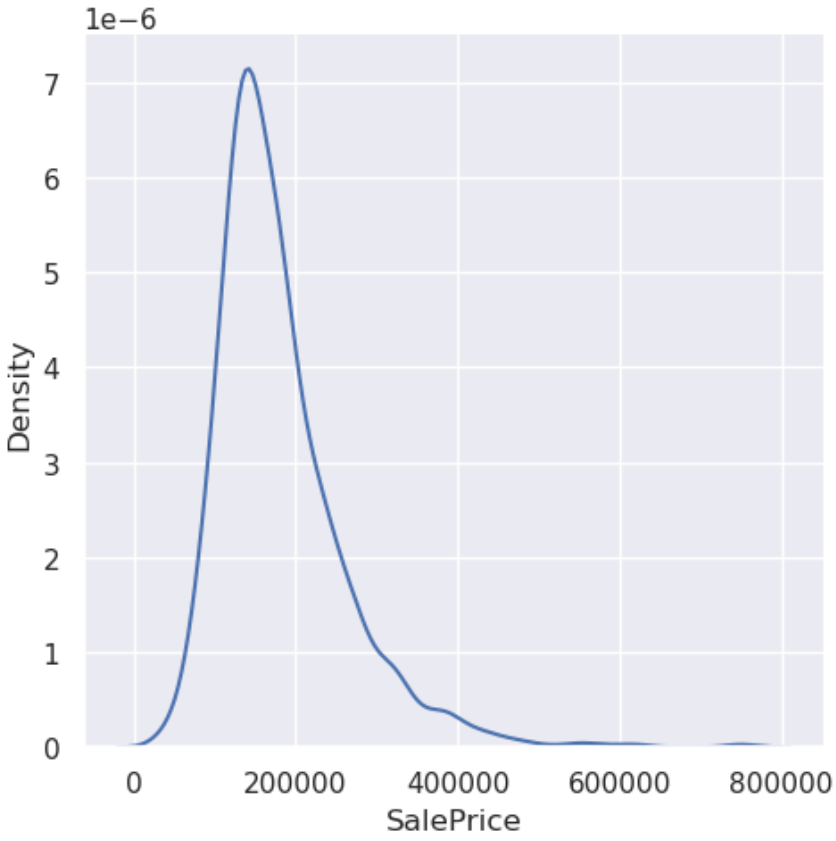
```
sb.violinplot(data["SalePrice"], orient="h")

# Rotate the x-axis labels by 45 degrees to the right
plt.xticks(rotation=45, ha="right")
```



18.17.5 Distribution plot

- The `sb.displot` function is a generic function that plots all distribution-related graphs.
- By default, it will plot a **histogram** (`kind=hist`).
- You can change the graph it plots by passing a `kind` argument to the function.
- The example below plots a **kernel density plot**.

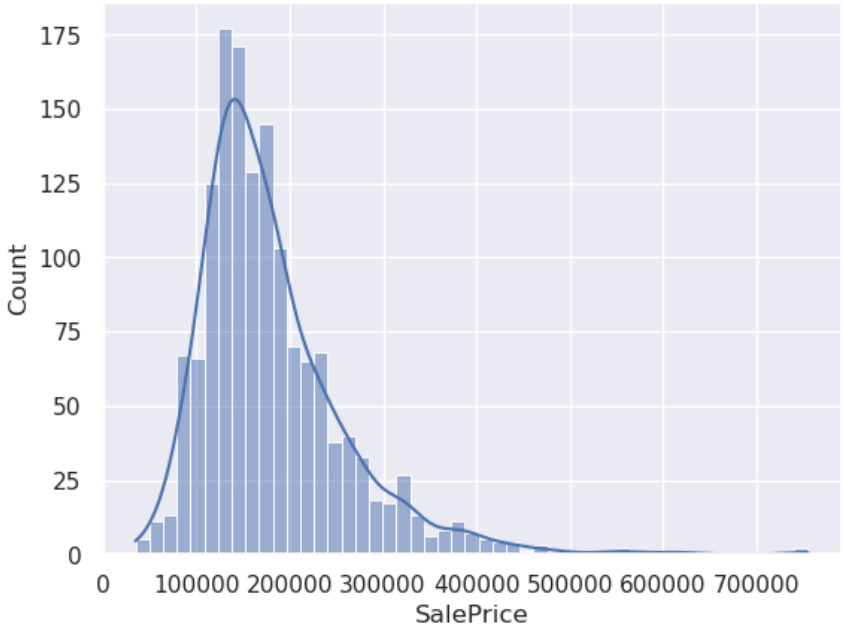
```
sb.displot(data["SalePrice"], kind="kde")
```



18.17.6 Histogram

- The histogram shows a number of bars indicating the value of a variable.
- The `kde` argument tells `seaborn` whether to include the kernel density plot together with the histogram plot. By default, it is set to `false`.

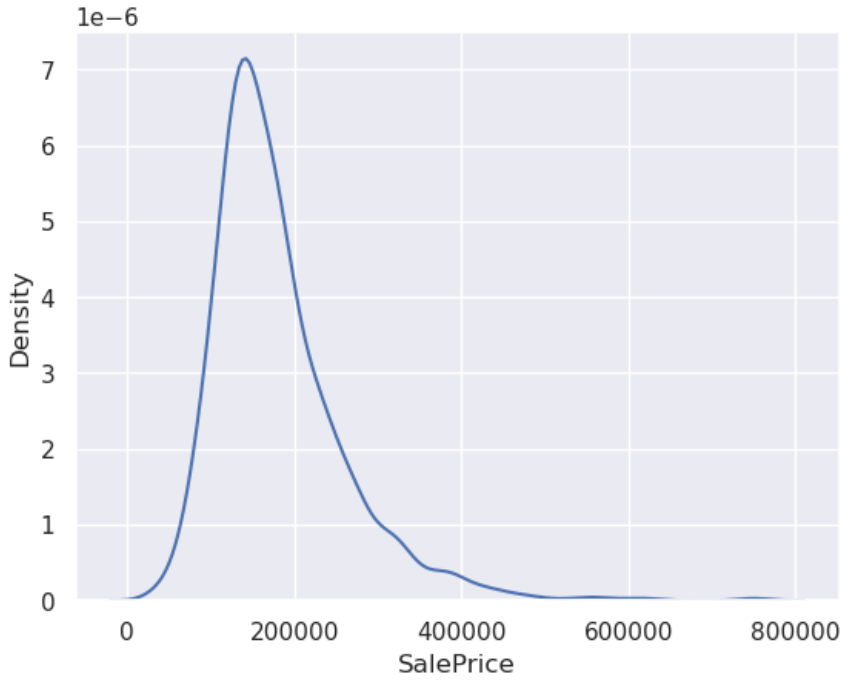
```
sb.histplot(data["SalePrice"], kde=True)
```



18.17.7 Kernel density plot

The kernel density plot draws a mountain-shaped graph that looks somewhat like the normal distribution.

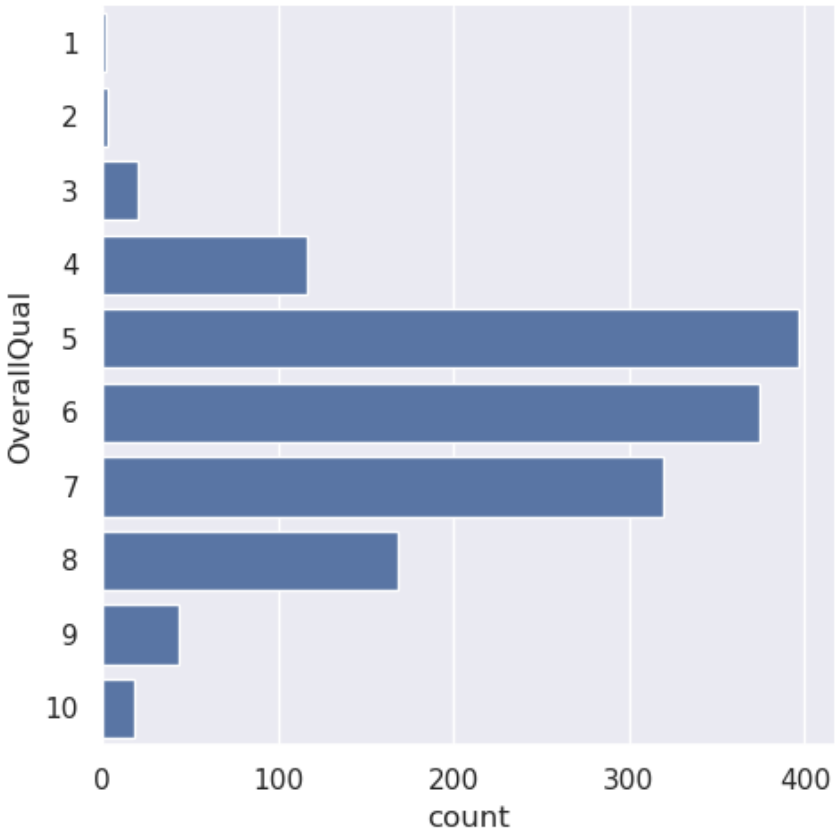
```
sb.kdeplot(data["SalePrice"])
```



18.17.8 Categorical plot

- The `sb.catplot` function is a generic function that plots all categorical related graphs.
- By default, it will plot a **strip plot**.
- You can change the graph it plots by passing a `kind` argument to the function.
- The example below plots a **count plot**.

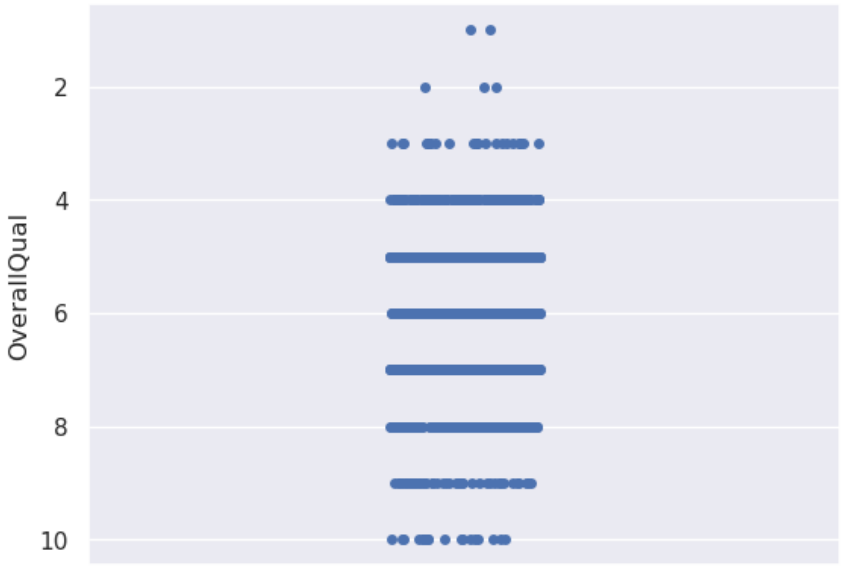
```
sb.catplot(categorical_data["OverallQual"], kind="count")
```



18.17.9 Strip plot

A strip plot lays out all the data points in strips, going either horizontally or vertically.

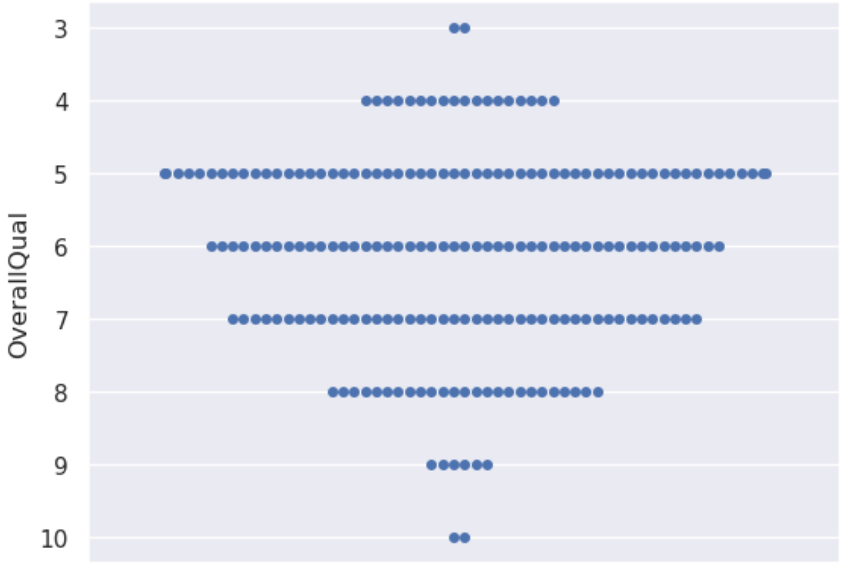
```
sb.stripplot(categorical_data["OverallQual"])
```



18.17.10 Swarm plot

A swarm plot is very similar to a strip plot, but the points are adjusted such that they don't overlap, so that you can see all points.

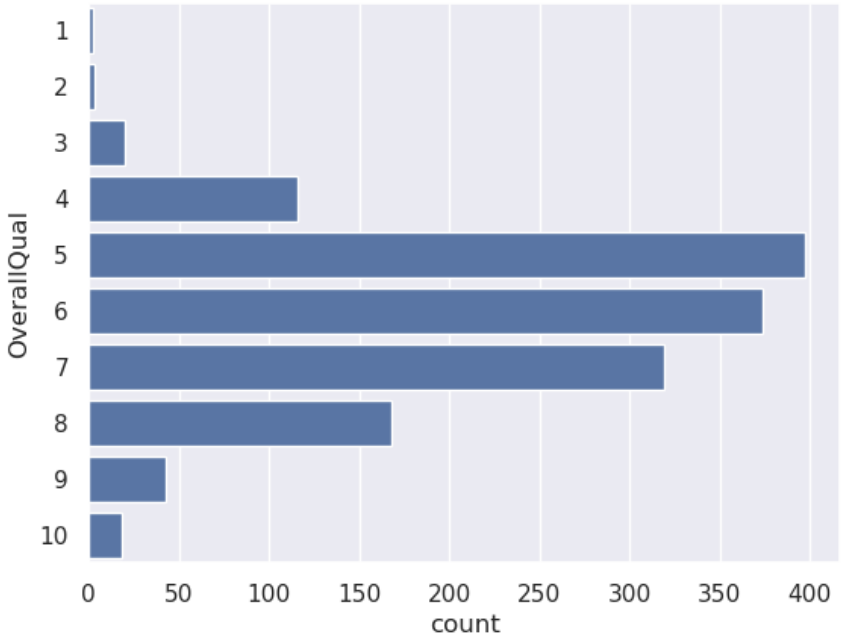
```
sb.swarmplot(categorical_data["OverallQual"][:200])
```



18.17.11 Count plot

A count plot shows the count of items in that category.

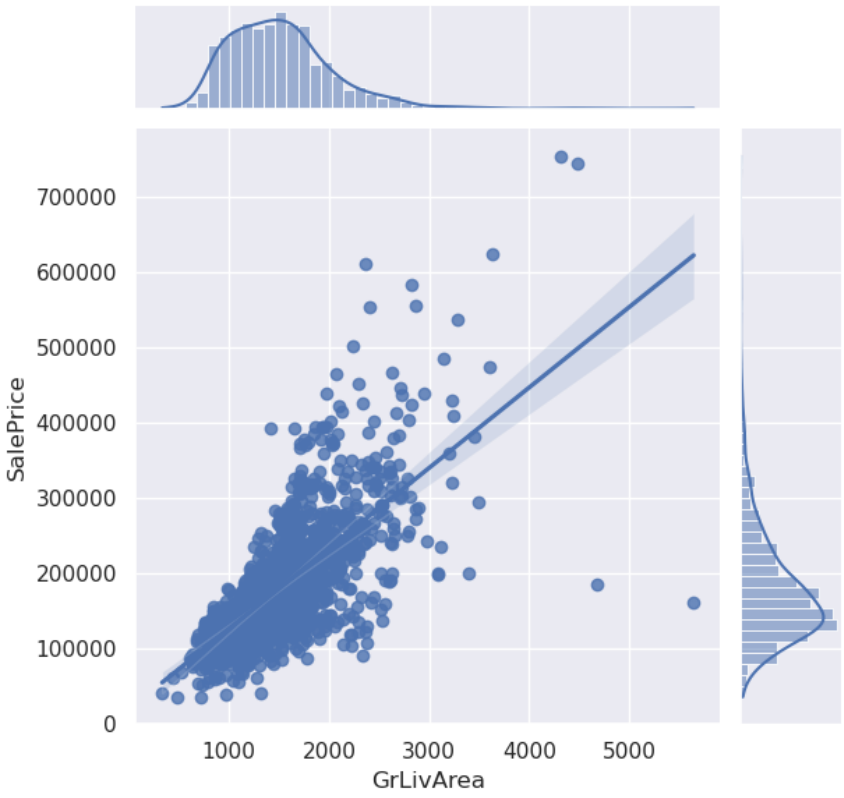
```
sb.countplot(categorical_data["OverallQual"])
```



18.17.12 Joint plot

Joint plots are for two sets of data, or for bi-variate graph plotting.

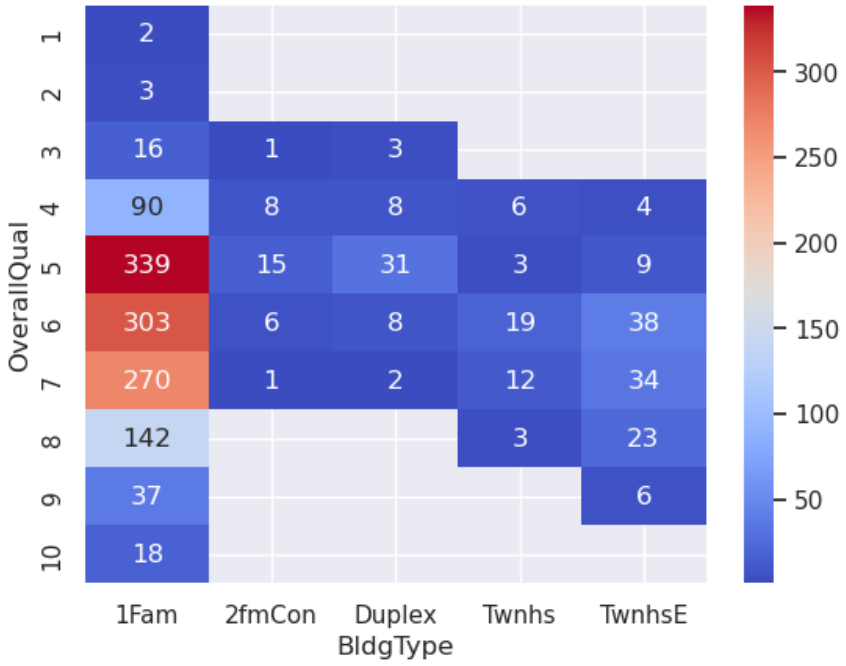
```
sb.jointplot(
    x="GrLivArea",
    y="SalePrice",
    data=data,

    # Add a linear regression line to the plot
    kind="reg",

    # Specify the height of the graph
    height=6,
)
```



18.17.13 Heatmaps

- Heatmaps are usually used to plot correlations, confusion matrices, or categorical data.
- You can view all the possible colour maps here.

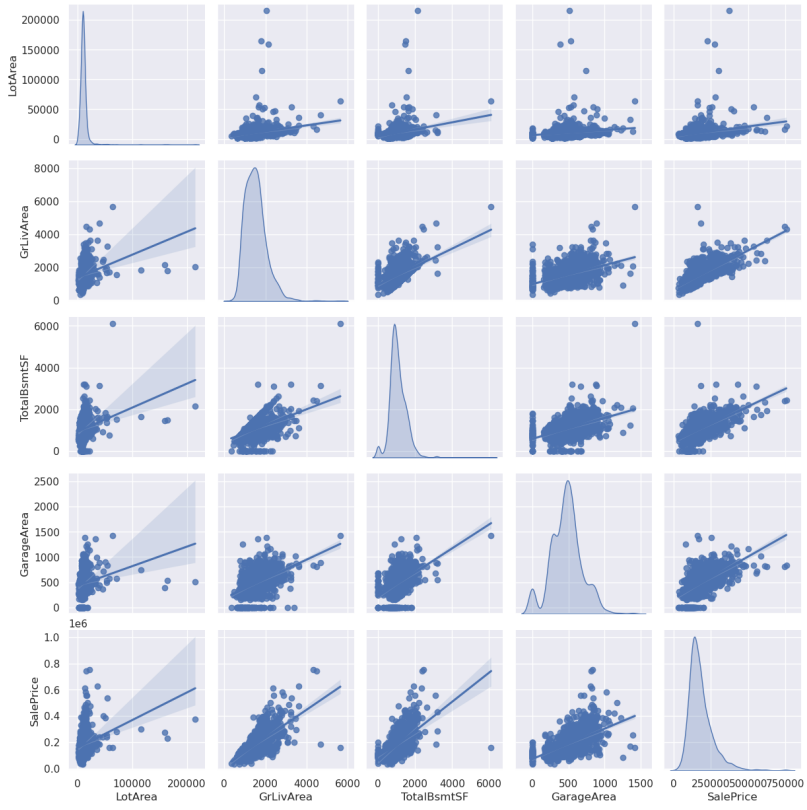
```
sb.heatmap(
    grouped_data,

    # Annotate the data
    annot=True,

    # Format the numbers as integers
    fmt=".0f",

    # Specify a colour map
    cmap="coolwarm",

    # Draw a colour bar
    cbar=True,
)
```



18.17.14 Pair plots

- The `sb.pairplot` function is for displaying the relationships between multiple variables at once.
- This function can only be used with numeric data as it plots the distributions of the variables.
- By default, the function will plot a scatter graph (`kind="scatter"`) of the variables, and the plots in the diagonal are automatically determined (`diag_kind="auto"`).
- The plots can be customised by setting `kind` to either `scatter` (default), `kde`, `hist`, or `reg`.
- The plot in the diagonal can be customised by setting `diag_kind` to either `auto` (default), `kde`, `hist`, `reg`.

```
sb.pairplot(data=numeric_data, diag_kind="kde", kind="reg")
```



18.17.15 Scatter plots

- Scatter plots are usually a way to visualise the distribution of data on a plane.
- You can give the function a colour to colour all the plotted points.

```
plt.figure(figsize=(16, 8))
plt.scatter(data["GrLivArea"], data["SalePrice"], color="red")
```