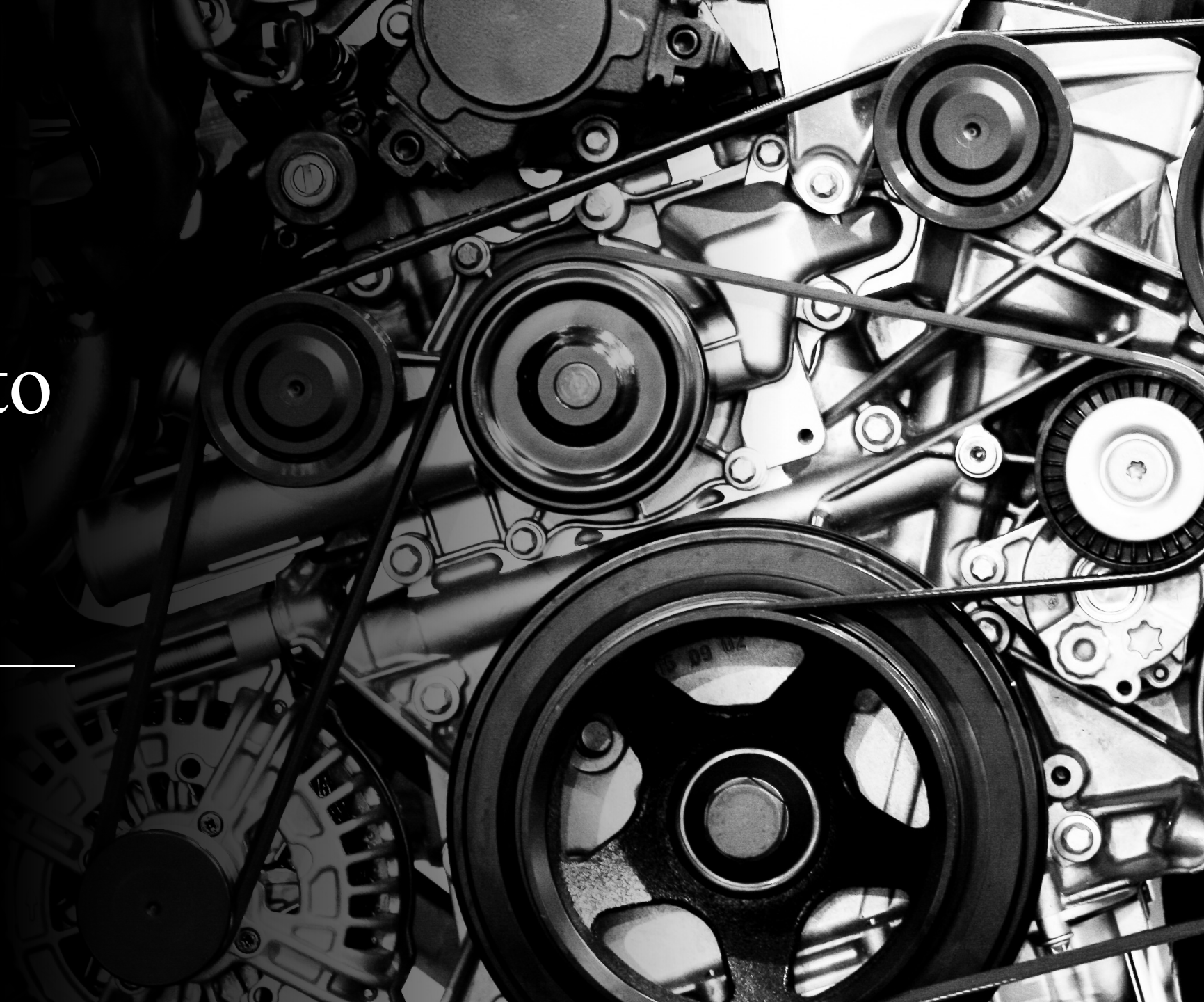
Introduction to Machine Learning



Introduction

What is machine learning?

- Machine learning is a way for computers to learn from examples, just like how kids learn by seeing and doing things
- Machine learning is a type of artificial intelligence where computer programs are designed to learn and improve from data and experience, without being explicitly programmed.
- In traditional computer programming, a programmer writes code that specifies how a program should behave in different situations. But with machine learning, instead of programming rules for how to solve a particular problem, a computer program is trained on a large dataset to learn patterns and relationships within the data.
- Once the program has learned these patterns, it can apply them to new data and make predictions or take actions based on what it has learned.

Introduction

What is machine learning? (Cont.)

- For example, a machine learning program can be trained on a large set of images of cats and dogs to learn how to recognize the differences between them.
- Once the program has learned what a cat and dog look like, it can analyze new images and predict whether each image contains a cat or a dog.
- Machine learning is used in many areas, such as natural language processing, computer vision, recommendation systems, and predictive analytics.
- *When a learning machine improves its performance at a given task over time, without reprogramming, it can be said to have learned something. Learning: improvement of performance (P) with experience (E) at a given task (T)*

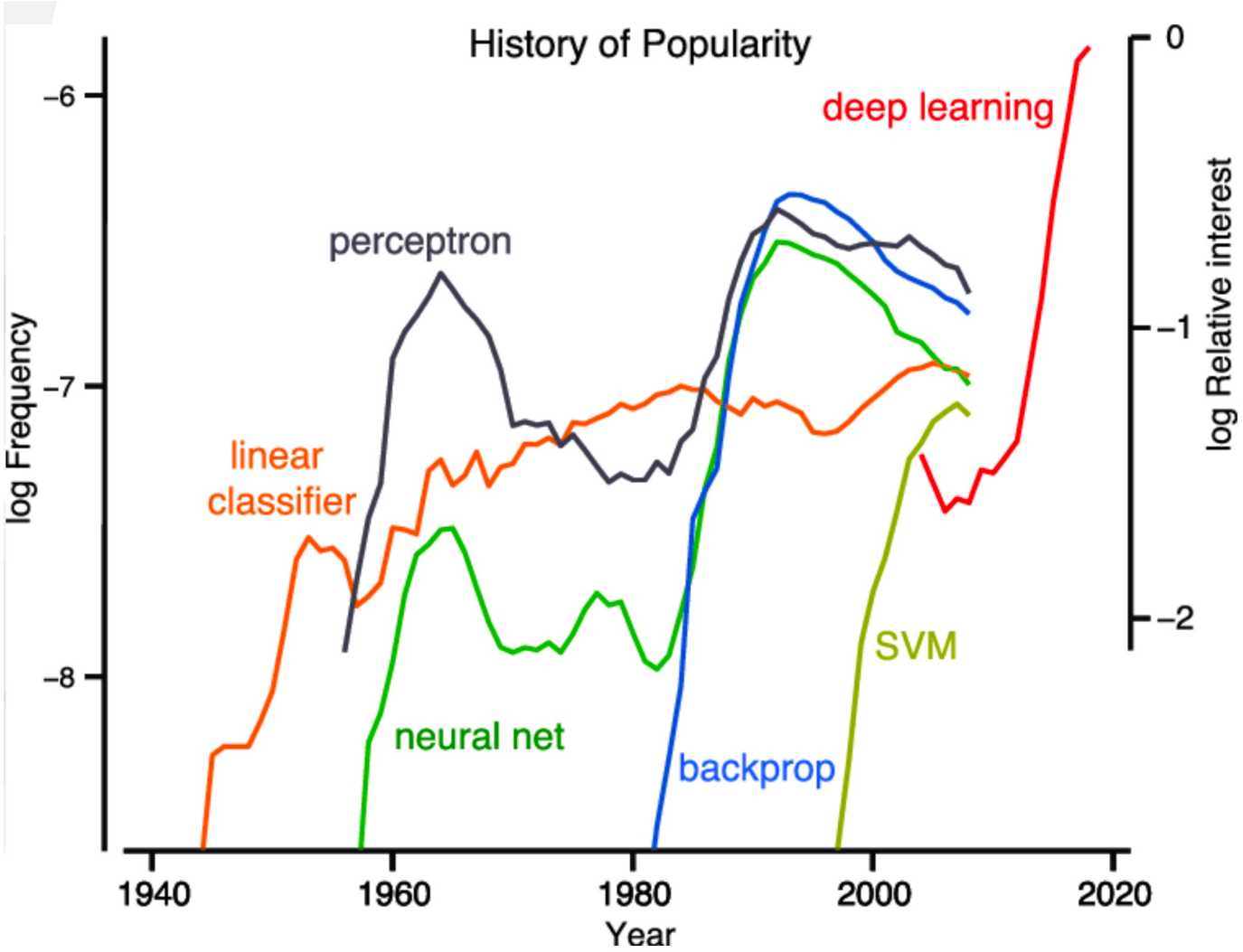
Introduction (Hist.)

Machine learning has a rich and interesting history that dates back to the mid-20th century. Here are some key milestones in the history of machine learning:

- *1943: Warren McCulloch and Walter Pitts published the first paper on artificial neurons, which laid the foundation for neural networks.*
- *1956: The term "artificial intelligence" was first coined by John McCarthy at the Dartmouth Conference.*
- *1959: Arthur Samuel coined the term "machine learning" and created the first computer program to play checkers.*
- *1960s-1980s: The field of machine learning went through a "winter" where funding and interest dwindled.*
- *1986: Geoffrey Hinton, David Rumelhart, and Ronald Williams published a paper on backpropagation, a widely used algorithm for training neural networks.*
- *1995: Vladimir Vapnik and Corinna Cortes developed the support vector machine (SVM) algorithm.*
- *2006: Geoffrey Hinton introduced deep learning, a method of training neural networks with multiple layers.*
- *2011: The Watson computer system, developed by IBM, defeated human champions on the game show Jeopardy!.*
- *2012: The ImageNet competition was won by a deep learning model for the first time, marking a breakthrough in computer vision.*
- *2016: AlphaGo, a computer program developed by DeepMind, defeated the world champion in the board game Go.*

Since then, machine learning has seen exponential growth and has become an integral part of many industries, including healthcare, finance, and transportation.

Introduction (Hist.)



Statistical vs Classical ML

- The terms "statistical machine learning" and "classical machine learning" are often used interchangeably, but they do have some key differences.
- Statistical machine learning emphasizes the use of statistical methods for modeling and prediction, while classical machine learning focuses on the development of algorithms and computational methods for learning from data. Statistical machine learning typically involves a more rigorous statistical analysis of the data and the use of probability theory and inference, while classical machine learning is more concerned with optimizing algorithmic performance.
- Another way to differentiate statistical and classical machine learning is by looking at their historical origins. Statistical machine learning has its roots in the field of statistics, with many of its foundational methods and concepts coming from statistical theory. In contrast, classical machine learning has its origins in computer science and engineering, with a focus on developing computational algorithms for solving problems.
- Despite these differences, statistical and classical machine learning are often used in similar contexts and can be applied to similar problems. Both approaches are used extensively in industry and academia and have contributed to the success of many real-world applications of machine learning.

Statistical vs Classical ML

Statistical Machine Learning Methods:

- *Linear regression*
- *Logistic regression*
- *Bayesian networks*
- *Gaussian processes*
- *Hidden Markov models*
- *Support vector machines (SVM) {overlap}*
- *Principal component analysis (PCA)*
- *Markov chain Monte Carlo (MCMC) methods*
- *Expectation-maximization (EM) algorithm {overlap}*
- *Naive Bayes classifier*

Classical Machine Learning Methods:

- *Decision trees*
- *Random forests*
- *K-nearest neighbours (KNN)*
- *Artificial neural networks (ANN)*
- *Convolutional neural networks (CNN)*
- *Recurrent neural networks (RNN)*
- *Gradient boosting*
- *Deep learning*
- *Clustering (e.g. k-means clustering)*
- *Association rules (e.g. Apriori algorithm)*
- *Reinforcement learning*

Modern Machine Learning

- *We are the last generation to be intellectually alone...we are the first to be intellectually together...we stand both before and after – Ray Kurzweil "The Age of Spiritual Machines - 1990*

- Modern ML involves using advanced algorithms and statistical models to allow machines to improve at performing specific tasks by learning from data without being explicitly programmed. It includes various techniques such as deep learning, neural networks, and natural language processing. Some examples of modern ML methods are:

    i. *Deep Learning: A subset of machine learning that involves training neural networks with multiple hidden layers to recognize patterns in large amounts of data. Applications include image recognition, natural language processing, and speech recognition.*

    ii. *Convolutional Neural Networks (CNNs): A type of neural network that is commonly used for image recognition and processing.*

    iii. *Recurrent Neural Networks (RNNs): A type of neural network that is useful for processing sequential data, such as natural language text or time-series data.*

    iv. *Generative Adversarial Networks (GANs): A type of neural network architecture that consists of two networks that work together to generate new, realistic data that resembles the training data.*

    v. *Reinforcement Learning: A type of machine learning that involves training an agent to interact with an environment by receiving rewards or punishments for certain actions. This approach is used in fields such as robotics and game AI.*

    vi. *Transfer Learning: A technique that involves using pre-trained models as a starting point for a new task. This approach can save time and resources, as well as improve performance.*

    vii. *Autoencoders: A type of neural network architecture that is used for unsupervised learning and feature extraction.*

- These are just a few examples of the modern ML methods that are being developed and applied today. As the field continues to evolve, new techniques and approaches are constantly being developed and refined.

Introduction (Applications)

1. Agriculture: ML is being used to improve crop yields and predict weather patterns, which can help farmers make more informed decisions about when to plant and harvest crops.
2. Healthcare: ML is being used to improve disease diagnosis, treatment and prevention in Africa. It can also help to identify and monitor outbreaks of diseases.
3. Finance: ML is being used to identify fraudulent transactions and to analyze customer data for better targeting of financial products and services.
4. Education: ML is being used to personalize learning and to provide customized feedback to students.
5. Energy: ML is being used to optimize the distribution and generation of electricity, which can help to reduce costs and improve reliability.
6. Transportation: ML is being used to improve traffic flow and to optimize public transportation routes.
7. Natural resource management: ML is being used to analyze satellite data to monitor deforestation, water resources, and wildlife conservation.

Some interesting developments 1700s to 2000s

Early Years		
1763	Bayes Theorem	Thomas Bayes
1913	Markov Chains	Andrey Markov
1950	Turing Machine	Alan Turing
1951	First Neural Networks	Marvin Minsky & Dean Edmond
1957	Perceptron	Frank Rosenblatt
1967	Nearest Neighbor	
1969	Limitations of Neural Networks	Marvin Minsky & Seymour Papert
1970	Backpropagation-Automatic Differentiation (AD)	Seppo Linnainmaa
1979	Neocognition – later inspired Convolution Neural Networks	Kunihiko Fukushima
1982	Recurrent Neural Networks- Hopfield Networks	John Hopfield

Some interesting developments 1700s to 2000s

Mid Years		
1985	NetTalk-basic automated word pronunciation	Terry Sejnowski
1989	Reinforcement Learning – Q- learning	Christopher Watkins
1989	Commercialization of ML	Evolver Project by Axcelis Inc.
1995	Random Forest	Tin Kam
1995	Support Vector Machines	Corinna Cortes & Vladimir Vapnik
1997	IBM Deep Blue Beats Kaparov	
1997	LSTM – improve efficiency on recurrent neural networks	Sepp Hochreiter & Jurgen Schmidhumber
1998	MNIT Database – for handwriting recognition	Yann LeCun & Team

Some interesting developments 1700s to 2000s

Modern Era		
2002	Torch ML Library release	
2011	IBM Watson beats humans in Jeopardy	
2012	Google Brain Algorithm manages to recognize cats on YouTube	Google Brain
2014	DeepFace – A neural network driven project with 97.35% accuracy on face recognition	Facebook
2016	AlphaGo beats professional human players	
2018	Attention is all you need paper	Vaswani et. al
2019	Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context	Dai et. al
2020	GPT-3	Open AI
2021	CLIP (Contrastive Language-Image Pre-Training), DALL-E	Open AI

End

Statistical vs Classical

Statistical vs Classical ML

- The terms "statistical machine learning" and "classical machine learning" are often used interchangeably, but they do have some key differences.
- Statistical machine learning emphasizes the use of statistical methods for modeling and prediction, while classical machine learning focuses on the development of algorithms and computational methods for learning from data. Statistical machine learning typically involves a more rigorous statistical analysis of the data and the use of probability theory and inference, while classical machine learning is more concerned with optimizing algorithmic performance.
- Another way to differentiate statistical and classical machine learning is by looking at their historical origins. Statistical machine learning has its roots in the field of statistics, with many of its foundational methods and concepts coming from statistical theory. In contrast, classical machine learning has its origins in computer science and engineering, with a focus on developing computational algorithms for solving problems.
- Despite these differences, *statistical and classical machine learning are often used in similar contexts and can be applied to similar problems*. Both approaches are used extensively in industry and academia and have contributed to the success of many real-world applications of machine learning.

Statistical vs Classical ML

Statistical Machine Learning Methods:

- *Linear regression*
- *Logistic regression*
- *Bayesian networks*
- *Gaussian processes*
- *Hidden Markov models*
- *Support vector machines (SVM) {overlap}*
- *Principal component analysis (PCA)*
- *Markov chain Monte Carlo (MCMC) methods*
- *Expectation-maximization (EM) algorithm {overlap}*
- *Naive Bayes classifier*

Classical Machine Learning Methods:

- *Decision trees*
- *Random forests*
- *K-nearest neighbours (KNN)*
- *Artificial neural networks (ANN)*
- *Convolutional neural networks (CNN)*
- *Recurrent neural networks (RNN)*
- *Gradient boosting*
- *Deep learning*
- *Clustering (e.g. k-means clustering)*
- *Association rules (e.g. Apriori algorithm)*
- *Reinforcement learning*

Example 1

- Statistical Machine Learning: In statistical machine learning, the focus is on estimating and modelling the underlying distribution of the data to make predictions.

- Here's an example of how to use a linear regression model from the scikit-learn library in Python to predict the price of a house based on its size:

```
from sklearn.linear_model import LinearRegression

# Create a linear regression model

model = LinearRegression()

# Train the model on the training data

model.fit(X_train, y_train)

# Predict the price of a house with a given size

size = 1500 price = model.predict([[size]])
```

Example 2

- In classical machine learning, the focus is on finding patterns in the data to make predictions, without making assumptions about the underlying distribution.

- Here's an example of how to use a decision tree classifier from the scikit-learn library in Python to predict whether a customer will buy a product based on their age and income:

```
from sklearn.tree import DecisionTreeClassifier

# Create a decision tree classifier
model = DecisionTreeClassifier()

# Train the model on the training data
model.fit(X_train, y_train)

# Predict whether a customer will buy a product based on their age and income
age = 30
income = 50000
will_buy = model.predict([[age, income]])
```

Modern Machine Learning

- *We are the last generation to be intellectually alone...we are the first to be intellectually together...we stand both before and after – Ray Kurzweil "The Age of Spiritual Machines - 1990*
- Modern ML involves using advanced algorithms and statistical models to allow machines to improve at performing specific tasks by learning from data without being explicitly programmed. It includes various techniques such as deep learning, neural networks, and natural language processing. Some examples of modern ML methods are:
  - *Deep Learning: A subset of machine learning that involves training neural networks with multiple hidden layers to recognize patterns in large amounts of data. Applications include image recognition, natural language processing, and speech recognition.*
  - *Convolutional Neural Networks (CNNs): A type of neural network that is commonly used for image recognition and processing.*
  - *Recurrent Neural Networks (RNNs): A type of neural network that is useful for processing sequential data, such as natural language text or time-series data.*

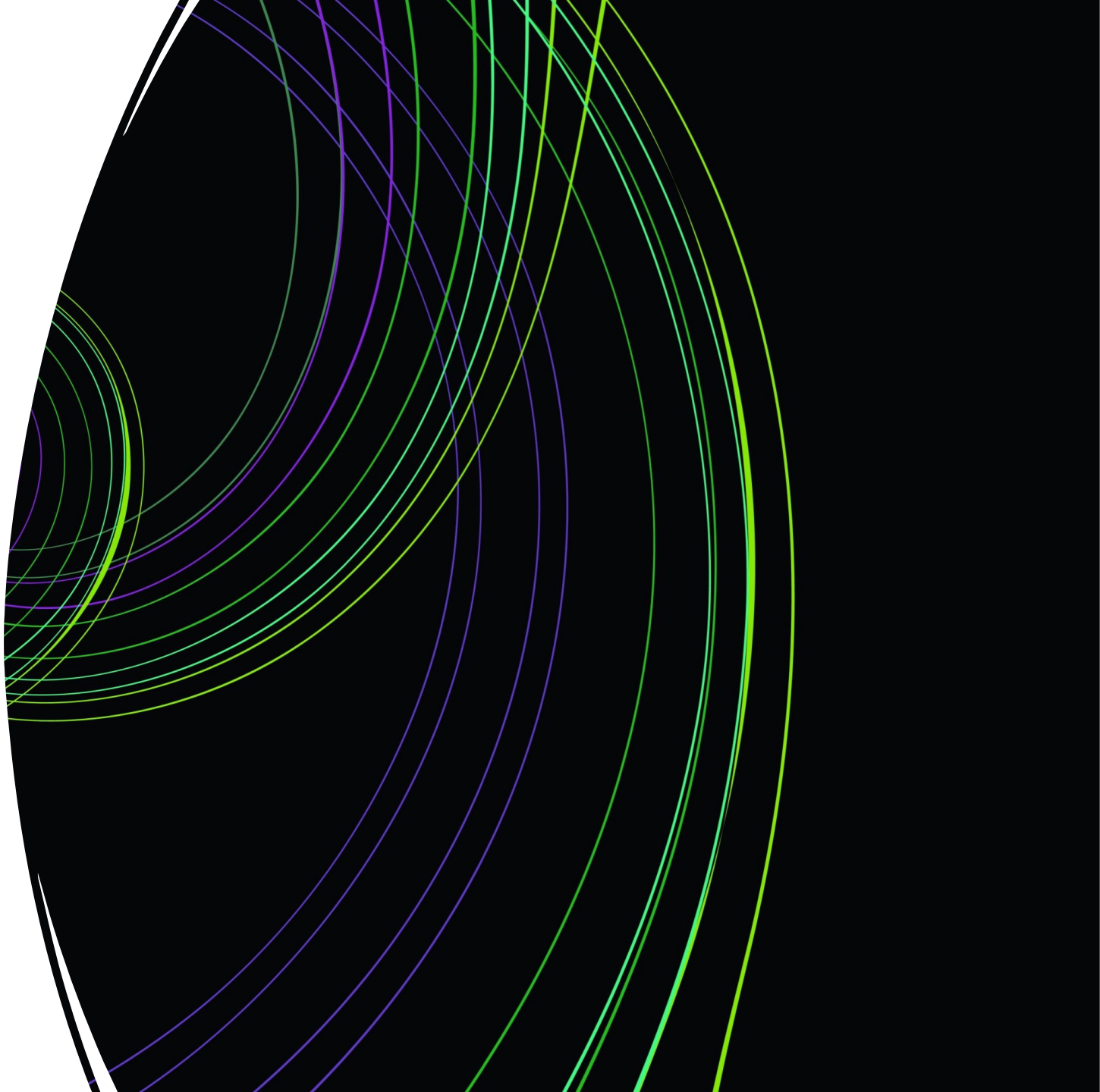
Modern Machine Learning

- *Generative Adversarial Networks (GANs): A type of neural network architecture that consists of two networks that work together to generate new, realistic data that resembles the training data.*
- *Reinforcement Learning: A type of machine learning that involves training an agent to interact with an environment by receiving rewards or punishments for certain actions. This approach is used in fields such as robotics and game AI.*
- *Transfer Learning: A technique that involves using pre-trained models as a starting point for a new task. This approach can save time and resources, as well as improve performance.*
- *Autoencoders: A type of neural network architecture that is used for unsupervised learning and feature extraction.*

- These are just a few examples of the modern ML methods that are being developed and applied today. As the field continues to evolve, new techniques and approaches are constantly being developed and refined.

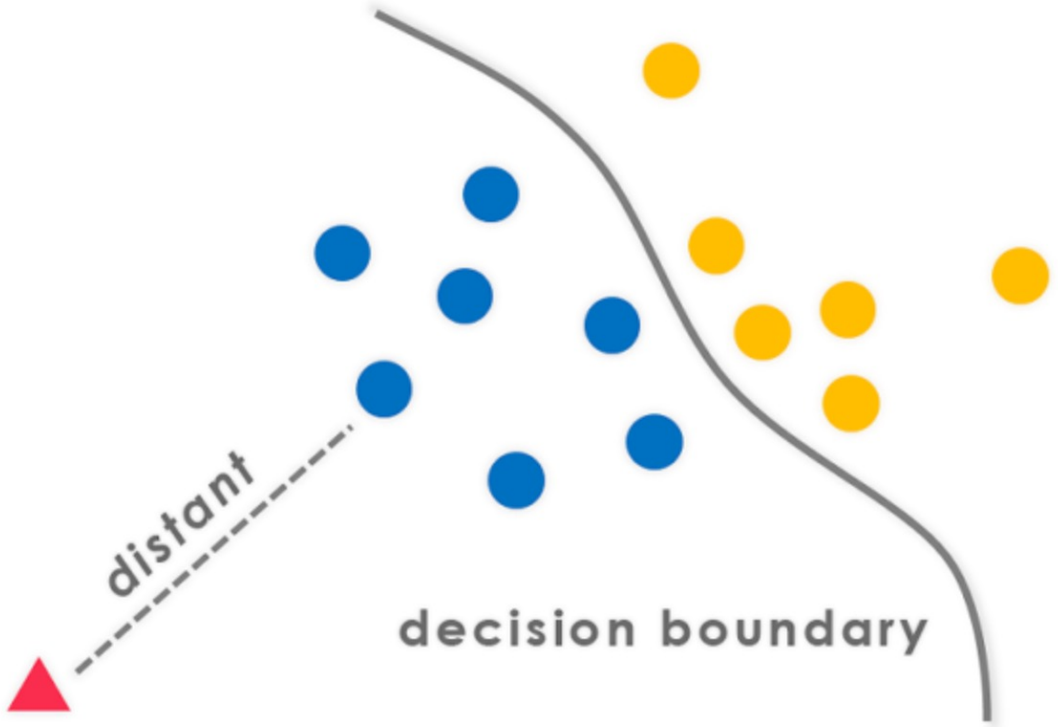
End

Discriminative vs Generative



Discriminative models

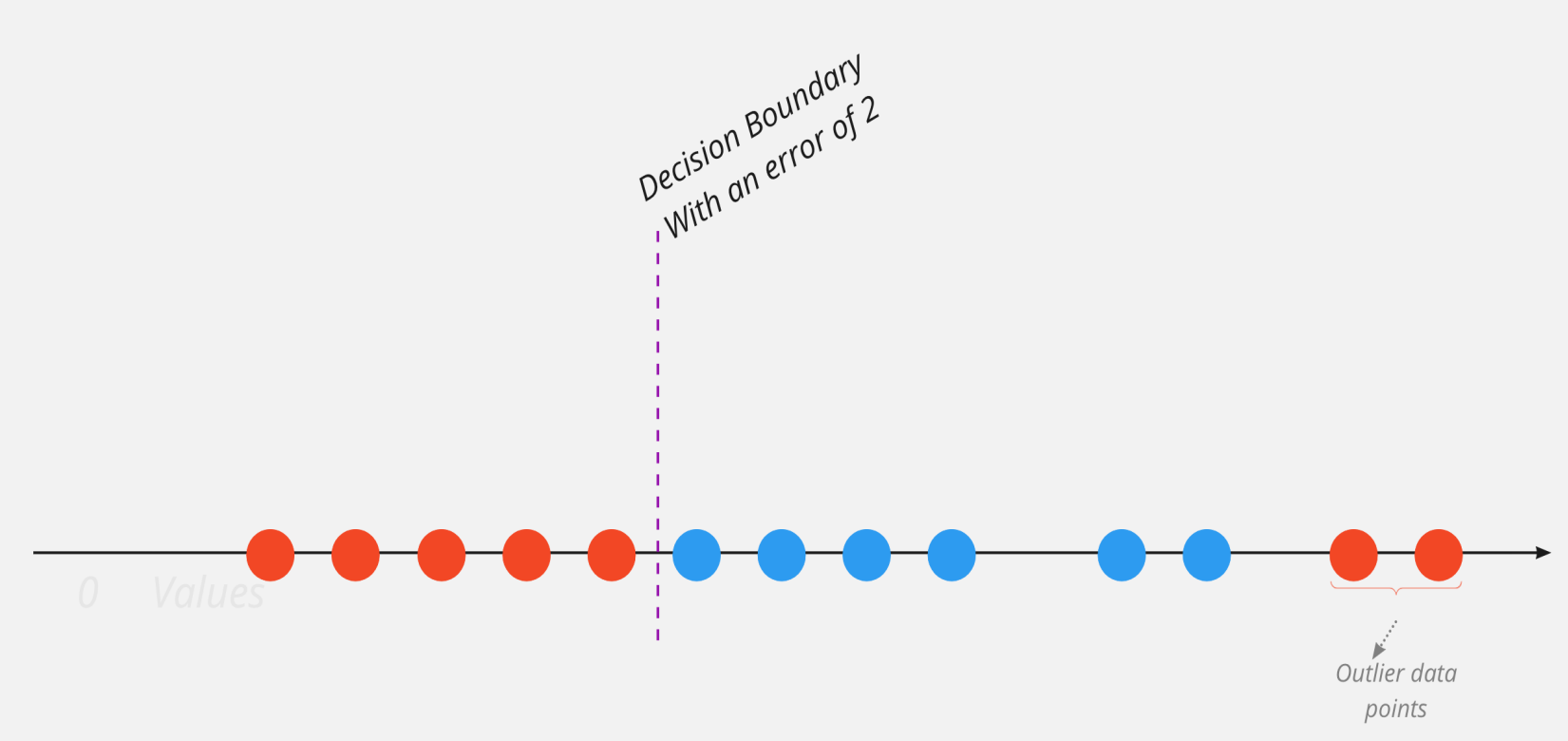
Focus on modeling the decision boundary between different classes of data.



Discriminative models

- The discriminative model refers to a class of models used in Statistical Classification, mainly used for supervised machine learning.
- These types of models are also known as conditional models since they learn the boundaries between classes or labels in a dataset.
- Discriminative models focus on modelling the decision boundary between classes in a classification problem. The goal is to learn a function that maps inputs to binary outputs, indicating the class label of the input.
- Discriminative models separate classes instead of modelling the conditional probability and don’t make any assumptions about the data points.
- But these models are not capable of generating new data points hence only usefull in separation

Discriminative Models *(Decision Boundary)*



- *Here we are presented with a set of 13 data points on the x axis. 7 for the red class and 6 for the blue class.*
- *A discriminative model aims to learn a decision boundary that separates the two classes.*
- *As much as there are two outlier on the far right, these will be considered as errors.*
- *We essentially have a decision boundary with an error of 2.*
- *The boundary isn't always a straight line, as some techniques in his category may generate curved line boundaries.*
- *They key goal is to find and optimal boundary.*

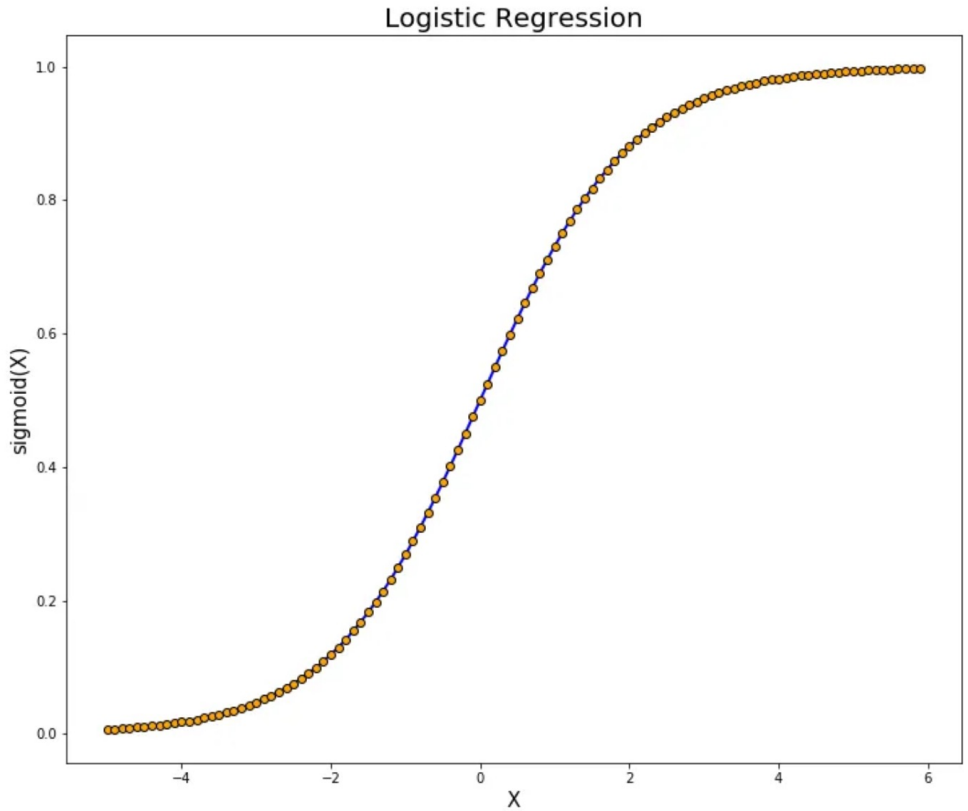
Discriminative Models

- Discriminative models are typically trained using supervised learning, where the model is provided with a labeled dataset of input features and output labels.
- The model is then trained to minimize a loss function that measures the difference between the predicted output labels and the true output labels.
- Discriminative models can be more effective than generative models when the focus is on accurate classification rather than generating new data or modeling the underlying probability distribution of the data.
- However, discriminative models can be more susceptible to overfitting than generative models, since they do not model the underlying distribution of the data.
- Discriminative models are widely used in many applications, including image classification, natural language processing, and speech recognition, among others.

Discriminative Models (Examples)

Logistic regression

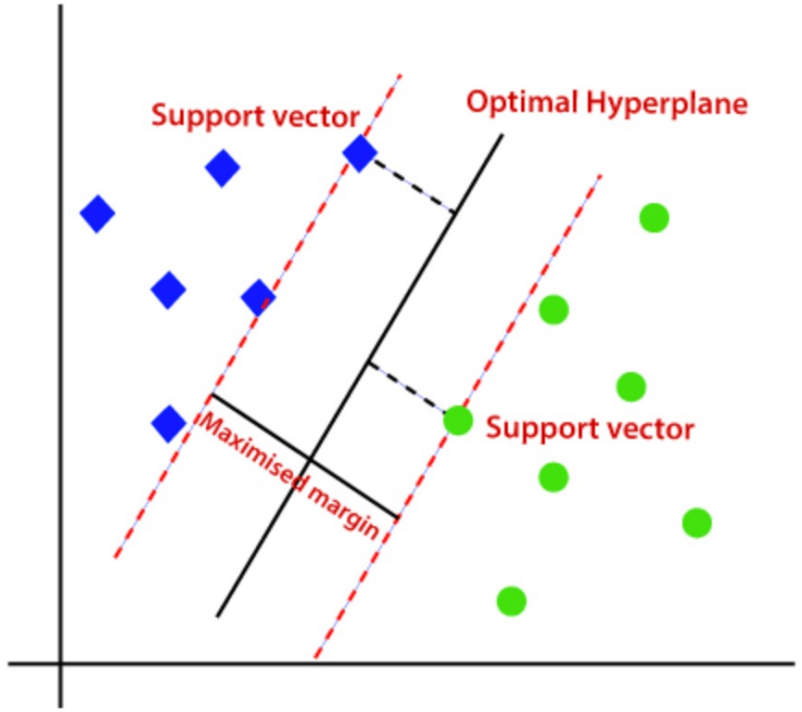
- A popular discriminative model that models the conditional probability of the output label given the input features directly.
- It learns a set of weights that are used to linearly combine the input features to produce a logit, which is then passed through a sigmoid function to obtain a probability estimate of the output label.



Discriminative Models (Examples)

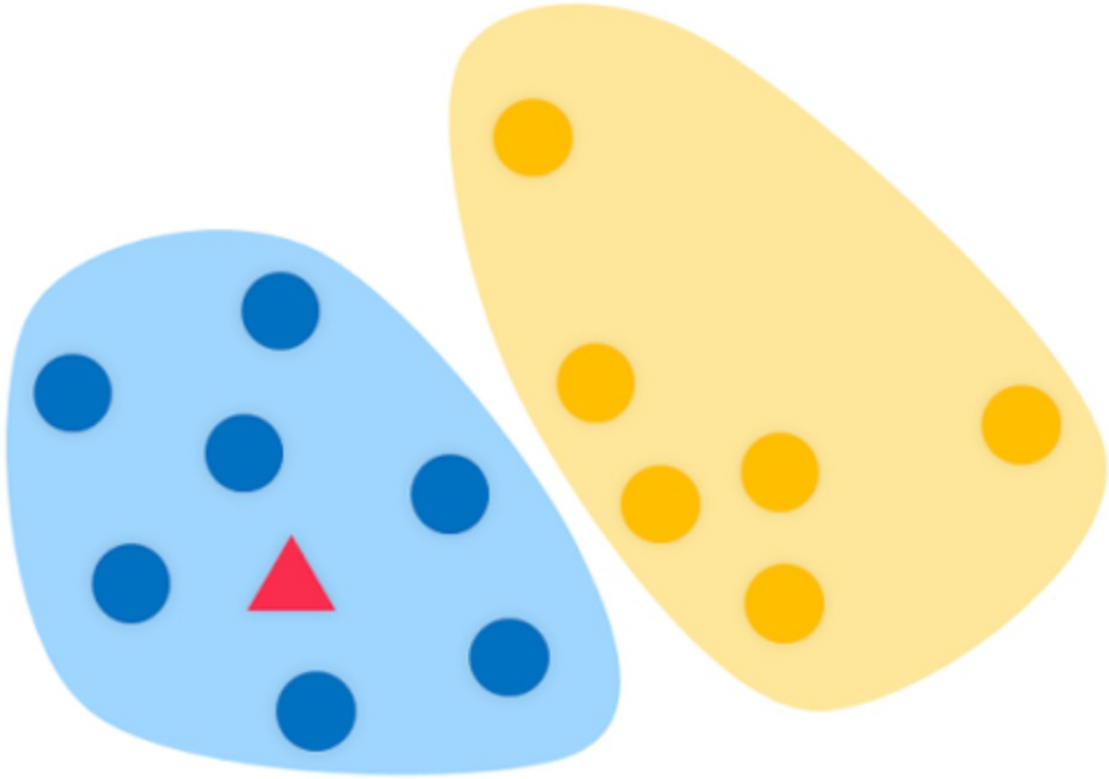
SVMs

- Another popular discriminative model that learn a hyperplane that separates the input data into different classes.
- The hyperplane is chosen to maximize the margin between the two classes, which is the distance between the hyperplane and the closest data points from each class.



Generative Models

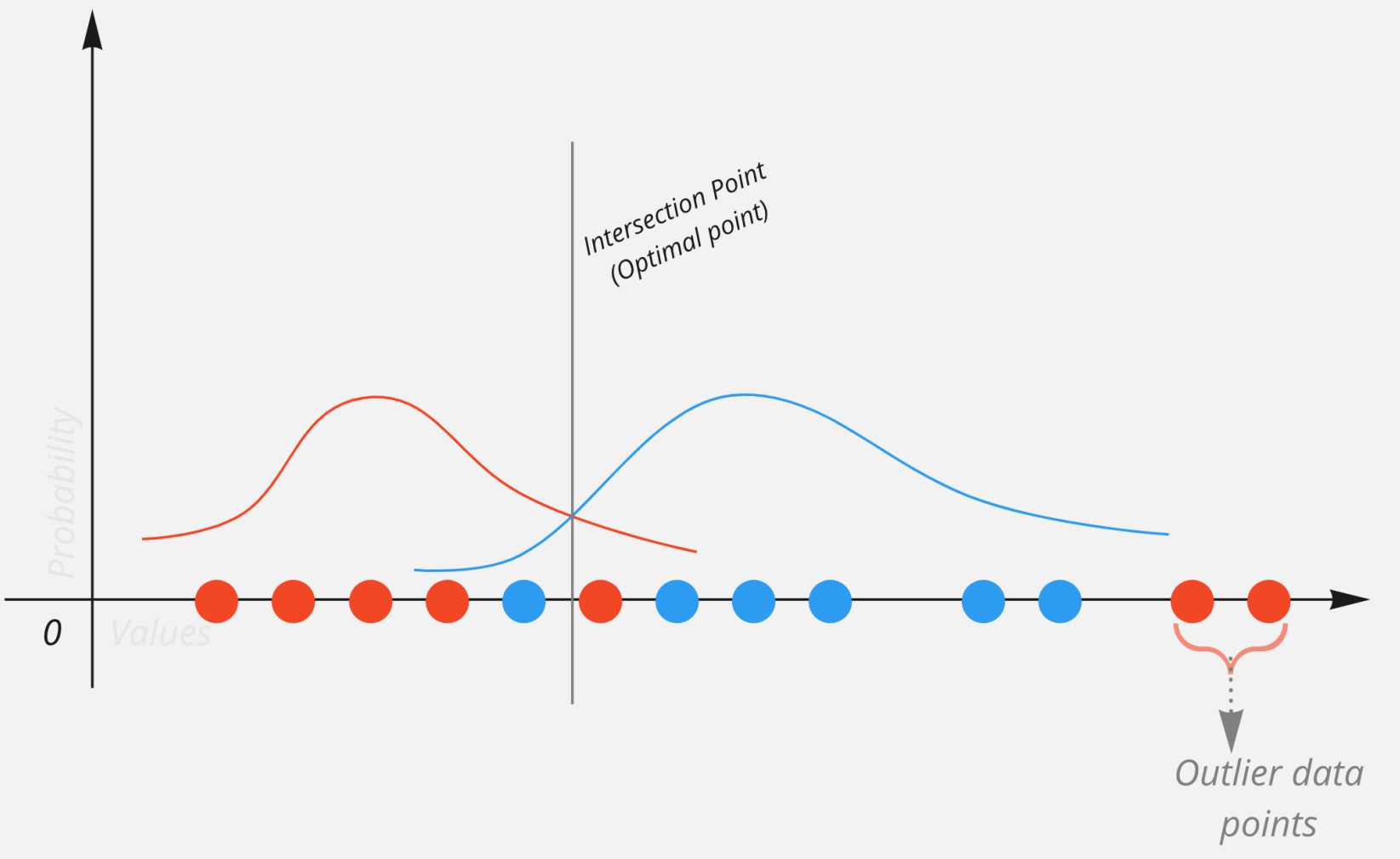
Aim to learn the underlying distribution of the data and use that to generate new samples.



Generative Models

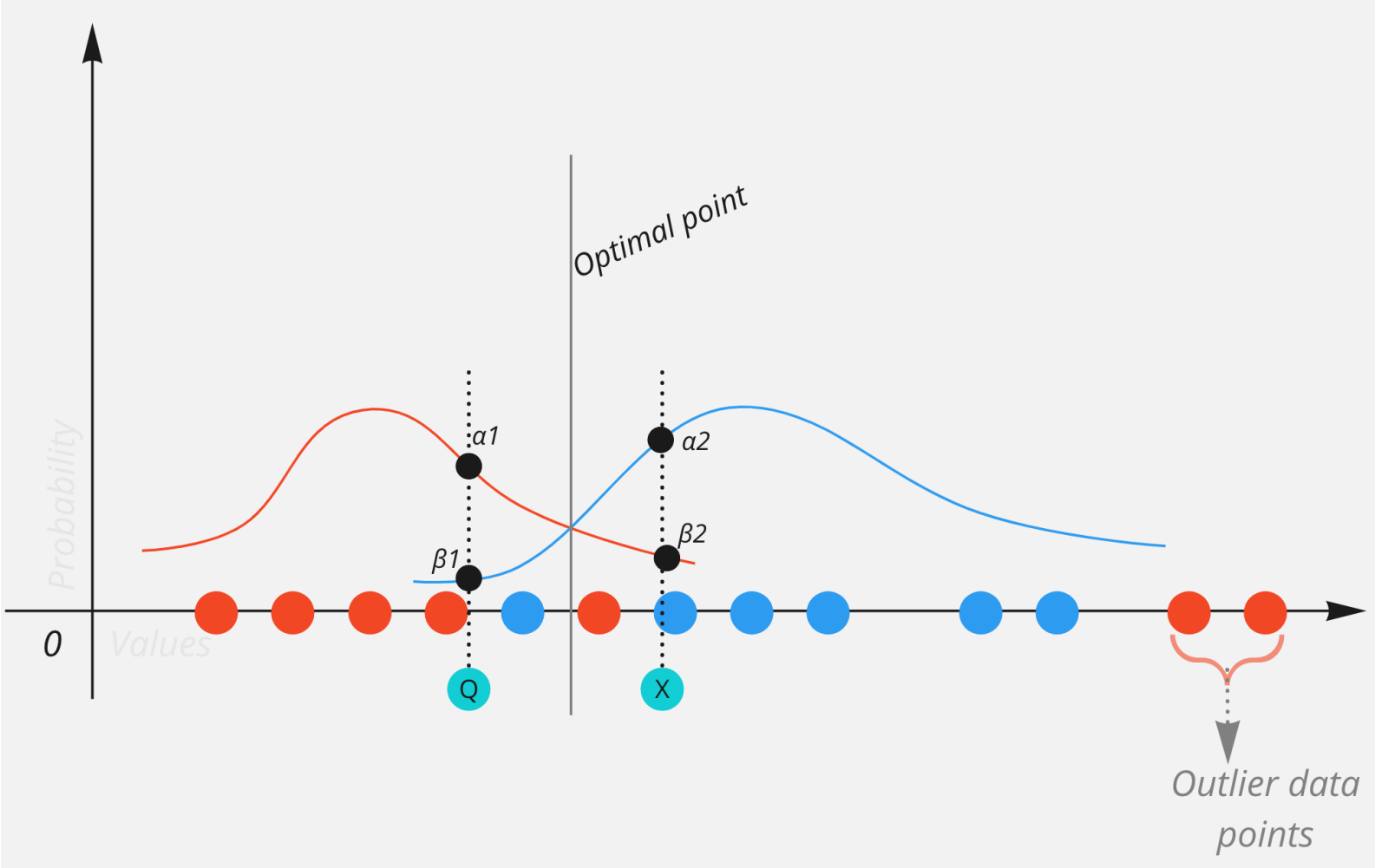
- Generative models are a type of machine learning model that model the underlying probability distribution of the data and can be used to generate new data that is similar to the training data.
- Generative models are considered a class of statistical models that can generate new data instances. These models are used in unsupervised machine learning as a means to perform tasks such as
    - Probability and Likelihood estimation,
    - Modeling data points
    - To describe the phenomenon in data,
    - To distinguish between classes based on these probabilities.
- Since these models often rely on the Bayes theorem to find the joint probability, generative models can tackle a more complex task than analogous discriminative models.

Generative Models *(Probability Logic Illustration)*



- *Here we are presented with a set of 13 data points on the x axis. 7 for the red class and 6 for the blue class.*
- *However, there are two red points on the far right, this are our outliers.*
- *An outlier is a data point that differs significantly from other observations. An outlier may be due to a variability in the measurement, an indication of novel data, or it may be the result of experimental error (From Wikipedia)*
- *Generative models assume there exists a normal distrbution within the dataset hence wont focus on the outliers within the dataset.*
- *As such, for classificiation, two normal distribution curves (for the two classes) can be used to illustrate the task.*
- *We identify an intersection point between the two curves and use this as our optimal classifcation point.*

Generative Models *(Probability Logic Illustration)*



- *We now introduce two new points that we wish to classify,* **Q** *and* **X.**
- *For Q, we can place it on the horizontal axis then generate a perpendicular line from it that intersects both distribution curves.*
- *This should give use two intersection points:* ***α1*** *and* ***β1***
- *We then proceed to assign it a class* ***(red / blue)*** *based on where the probability is high.*
- *We know this high probability point by comparing the two intersection point then picking the greater of the two.*
- *In this case* ***α1*** *is greater than* ***β1*** *hence we assign the new data point Q to belong to the red class.*

- *For X, we apply the exact logic and end up classify it as blue since the probability of blue is higher.*

Generative Models (Examples)

- **Markov Chain Monte Carlo (MCMC)** methods: MCMC methods are a class of algorithms that can generate samples from a probability distribution. They are commonly used in statistical physics and Bayesian inference.
- **Hidden Markov Models (HMMs):** HMMs are a type of statistical model that can generate sequences of observations. They are commonly used in speech recognition and natural language processing.
- **Gaussian Mixture Models (GMMs):** GMMs are a type of probabilistic model that can generate new data by sampling from a mixture of Gaussian distributions. They are commonly used in clustering and density estimation.
- **Variational Autoencoders (VAEs):** VAEs are a type of neural network that can learn to generate new data by compressing and decompressing existing data. They are commonly used for generating images and have been used for applications such as generating realistic human faces.
- **Generative Adversarial Networks (GANs):** GANs are a type of neural network that consists of two models - a generator and a discriminator - that are trained in a game-like manner to generate realistic data. They have been used for applications such as generating realistic images and videos.

Comparisons

Based on Performance

- Generative models need fewer data to train compared with discriminative models since generative models are more biased as they make stronger assumptions, i.e., assumption of conditional independence.

Based on Missing Data

- In general, if we have missing data in our dataset, then Generative models can work with these missing data, while discriminative models can’t. This is because, in generative models, we can still estimate the posterior by marginalizing the unseen variables. However, discriminative models usually require all the features X to be observed.

Based on the Accuracy Score

- If the assumption of conditional independence violates, then at that time, generative models are less accurate than discriminative models.

Based on Applications

- Discriminative models are called “discriminative” since they are useful for discriminating Y’s label, i.e., target outcome, so they can only solve classification problems. In contrast, Generative models have more applications besides classification, such as samplings, Bayes learning, MAP inference, etc.

End

Other Models:

- *Probabilistic Models*
- *Formal Rule System(s)*
- *State Machines*
- *Vector Space Models*
- *Dimension Reduction Models*

Probabilistic Models

- Probabilistic models incorporate uncertainty into the modeling process by assigning probabilities to different outcomes.
- They can be either discriminative or generative, but they explicitly model the probability distributions of the input data and/or output labels.
- Examples of probabilistic models include
  - *Bayesian networks,*
  - *Markov random fields (MRFs), and*
  - *Gaussian processes (GPs)*
- One of the main advantages of probabilistic models is that they can handle uncertainty and variability in data.
- For example, when working with real-world data, there is often noise or randomness that can affect the accuracy of predictions. Probabilistic models can help to capture this uncertainty by modelling the data as a probability distribution.

Probability Theory *(Principles)*

1. **Sample space:** The sample space is the set of all possible outcomes of an experiment or random event. Denoted by the symbol S.

2. **Event:** An event is a subset of the sample space. It represents a particular outcome or a set of outcomes that may occur in an experiment. Events become denoted by capital letters, such as A, B, or C.

3. **Probability**: Probability is a measure of the likelihood of an event occurring. Denoted by the symbol P(A) and ranges from 0 to 1, where 0 represents impossibility and 1 represents certainty.

4. **Additive rule:** The additive rule states that the probability of the union of two or more events is equal to the sum of their individual probabilities minus the probability of their intersection. That is, P(A or B) = P(A) + P(B) – P(A and B).

5. **Multiplicative rule:** The multiplicative rule states that the probability of the intersection of two or more independent events is equal to the product of their individual probabilities. That is, P(A and B) = P(A) x P(B).

6. **Conditional probability:** Conditional probability is the probability of an event A given that another event B has occurred. It is denoted by P(A | B) and is calculated as the probability of the intersection of A and B divided by the probability of B.

7. **Bayes' theorem:** Bayes' theorem is a formula for computing the probability of an event based on prior knowledge or information. It states that the probability of event A given event B is equal to the probability of event B given event A multiplied by the prior probability of event A divided by the prior probability of event B.

Formal Rule System(s)

- Formal rule systems are a class of machine learning models that are based on a set of predefined rules or logic statements. These models use *a set of if-then statements* to make decisions or predictions based on the input data.
- Formal rule systems are often used in expert systems, which are systems that mimic the decision-making capabilities of human experts in a particular domain.
- *Expert knowledge or domain-specific knowledge* is used to derive the rules for formal rule systems.
- These rules can be represented in various forms, such as *logical statements, decision trees, or other symbolic representations.* The objective of formal rule systems is to use these rules to make accurate predictions or decisions based on the input data.
- They offer an interpretable approach to machine learning that can be useful in expert systems or other applications where explain-ability is important. *Nevertheless, they may not be as flexible or adaptable as other machine learning alts.*

Formal Rule System(s)

- Formal rule systems have the advantage of being *transparent and explainable.*
- Since the rules used in these systems are explicitly defined, it is possible to trace the decision-making process and understand how the model arrived at a particular decision.
- This transparency can be useful in domains like healthcare or finance, where the reasoning behind a decision is critical.

- However, one disadvantage of formal rule systems is that they can be *inflexible and fragile.*
- If the rules are too specific or narrow, the model may not be able to handle new or unexpected situations.
- On the other hand, if the rules are too general or broad, they may not be able to make accurate predictions or decisions.

State Machines (*Also known as Finite State Machines*)

- Are based on a set of states and transitions between those states (Known number hence the name finite).
- These models are used to model systems that have a finite number of states and can be used to make decisions or predictions based on the input data.
- Each state represents a particular condition or situation, and the transitions between states represent the actions or events that can occur. The model uses the input data to determine the current state and then transitions to the next state based on the rules or logic of the model.
- These models provide a simple and interpretable approach to machine learning that can be useful in a variety of applications. However, they may not be suitable for all types of data or applications, and may require careful design and management to ensure accuracy and effectiveness.
- State machine models are often used in applications where the input data has a temporal or sequential nature, such as in *natural language processing*, *speech recognition*, and *control systems*. In these applications, there is need to track the sequence of events and make decisions or predictions based on that sequence.

State Machines

- State machine models is that they are very interpretable and can be used to model complex systems in a simple and easy-to-understand way.
- Because the states and transitions are explicitly defined, it is possible to understand how the model arrived at a particular decision or prediction.

- However these state machines can become very complex and difficult to manage as the number of states and transitions increases.
- Additionally, if the model is not designed properly, it may not be able to handle new or unexpected situations, or may not be able to make accurate predictions or decisions.

Vector Space Models (*VSMs*)

- Vector Space Models (VSMs) are a type of machine learning model used for natural language processing (NLP). These models represent words as vectors in a high-dimensional space, allowing them to be compared and manipulated mathematically.
- In VSMs, each word in a vocabulary is represented as a vector in a high-dimensional space, such as a 100, 200, or 300-dimensional space. The values in each dimension of the vector represent some characteristic or feature of the word, such as its frequency of use, semantic meaning, or contextual relationships with other words.
- Examples:
  - One common simple VSM is the Bag-of-Words (BoW) model, which represents a text as a vector of word frequencies. This model ignores word order and focuses only on the presence or absence of words in a document.
  - Another VSM is the Word2Vec model, which uses a neural network to learn vector representations of words based on their contexts in a large corpus of text. *(See Demo for Illustration)*

Vector Space Models (*VSMs*)

- VSMs are useful in various NLP tasks, such as document classification, information retrieval, and sentiment analysis.
  - In document classification, VSMs can be used to represent documents as vectors and classify them into pre-defined categories.
  - In information retrieval, VSMs can be used to match query terms with relevant documents based on their vector representations.
  - In sentiment analysis, VSMs can be used to classify the sentiment of a text based on the vector representations of its constituent words.
- One limitation of VSMs is the "curse of dimensionality," which refers to the difficulty in processing high-dimensional data.
- *As the number of dimensions increases, the amount of data needed to accurately represent the vectors increases exponentially, making computation and storage more difficult.*
- Therefore, various techniques such as *dimensionality reduction* and *sparsity regularization* are used to address this issue.

Dimension Reduction Models

- Dimensionality reduction is the process of reducing the number of features or variables in a dataset while retaining the maximum amount of useful information.
- It is a commonly used technique in machine learning and data science to improve the performance and efficiency of models by reducing the complexity of the data.
- ALT description: *It is used to overcome the curse of dimensionality, which refers to the problem of having too many input features in relation to the number of observations.*
- There are two main types of dimensionality reduction models:
    1. Feature selection
    2. Feature extraction
- The benefits include reduced computational complexity, improved model performance, and increased interpretability of the data. However, there are also some drawbacks, such as the potential loss of information and increased complexity in the model selection process.

Dimension Reduction (Types)

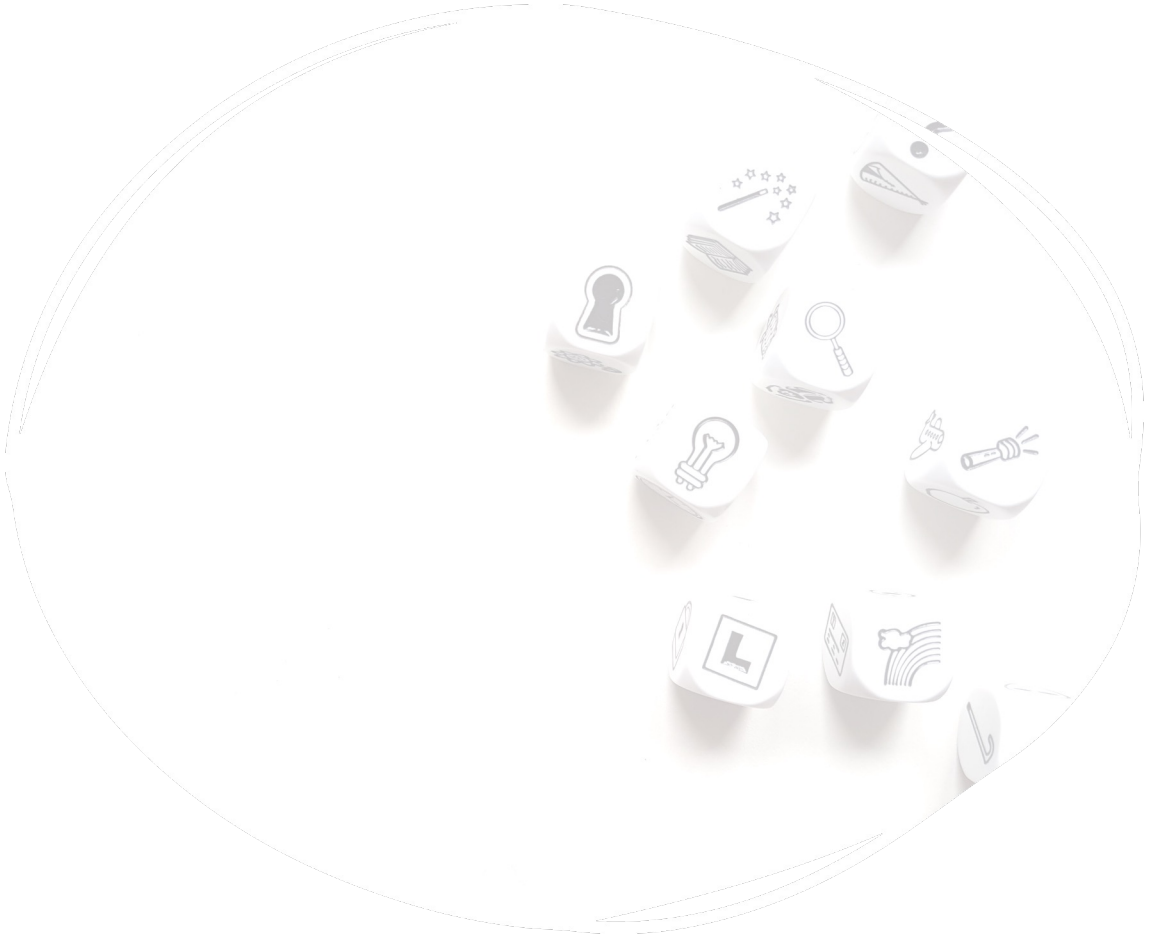
Feature selection

- Involves selecting a subset of the original features that are most relevant to the problem at hand. This can be done using various statistical or machine learning techniques, such as correlation analysis, mutual information, or recursive feature elimination.
- The resulting subset of features (From the original whole set) is then used for training the model, which can lead to improved accuracy, reduced overfitting, and faster training times.

Feature extraction

- Involves transforming the original features into a lower-dimensional space using mathematical techniques such as *principal component analysis* (PCA), *linear discriminant analysis* (LDA), or *t-distributed stochastic neighbor embedding* (t-SNE).
- This creates a set of new features that capture the most important information in the original data, while discarding irrelevant or redundant features. The resulting lower-dimensional representation can be used for visualization, clustering, classification, or other tasks.

End



Bayes' Theorem

Bayes' Theorem

- Bayes' theorem is a formula for computing the probability of an event based on prior knowledge or information.
- It states that *the probability of event A given event B is equal to the probability of event B given event A multiplied by the prior probability of event A divided by the prior probability of event B.*

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

Bayes' Theorem (Simpler Analogy)

- Bayes' Theorem is a way of thinking about how we update our beliefs when we learn new information.
- Imagine you are trying to figure out if it's going to rain tomorrow. You know that it usually doesn't rain in your city, but you also know that there's a chance it could rain. You don't know for sure, so you have a belief that there's a certain probability that it will rain tomorrow.
- Now, let's say your friend tells you that the weather forecast predicts a 50% chance of rain tomorrow. This is new information that you didn't have before. Bayes' Theorem tells you to update your belief based on this new information.
- In this case, your new belief will be somewhere between your original belief and your friend's belief. You might now believe that there's a higher chance of rain than you did before, but you might not believe it's as likely as your friend does.
- Bayes' Theorem is a way of *taking what you already know and combining it with new information to get a better understanding of what's going on.* It's a really powerful tool that scientists, engineers, and other people use to make better decisions and understand the world around them.

Bayes' Theorem (Example 1)

Assume we have a bag with 3 red balls and 2 blue balls. We want to know the probability of drawing a red ball from the bag.

Here's how we can use Bayes' Theorem to calculate the probability:

- P(R) is the prior probability of drawing a red ball, which is 3/5 because there are 3 red balls out of 5 total balls in the bag.
- P(B) is the prior probability of drawing a blue ball, which is 2/5.
- P(D) is the probability of drawing a ball from the bag, which is 1 because you will definitely draw a ball.
- P(D|R) is the probability of observing the data (drawing a ball) given that you drew a red ball, which is 1 because you will definitely observe a ball if you drew a red ball.
- P(D|B) is the probability of observing the data given that you drew a blue ball, which is also 1.
- P(R|D) is the posterior probability of drawing a red ball given that you observed a ball.

Bayes' Theorem (Example)

Now, let's apply Bayes' Theorem to calculate P(R|D):

- `# calculate the marginal likelihood of the data`
- `P(D) = P(D|R) * P(R) + P(D|B) * P(B) = 1`
- `# apply Bayes' Theorem`
- `P(R|D) = P(D|R) * P(R) / P(D) = 3/5 = 0.6`

- $P(R|D) = \frac{P(D|R) x\, P(R)}{P(D)}$
- $P(R|D) = \frac{1\, x \frac{3}{5}}{1}$
- $P(R|D) = 0.6$

Bayes' Theorem (Example 2)

Let's say we have a medical test for a disease (say Malaria) and we want to calculate the probability that a patient has the disease given a positive test result.

- P(A) is the prior probability of having the disease before the test is taken.
- P(B) is the probability of a positive test result.

Using Bayes' Theorem, we can calculate the posterior probability of having the disease given a positive test result:

$P(A|B) = P(B|A) * P(A) / P(B)$

- P(B|A) is the probability of a positive test result given that the patient has the disease. This is known as the sensitivity of the test.
- P(A) is the prior probability of having the disease, which is typically estimated based on the prevalence of the disease in the population.
- P(B) is the probability of a positive test result, which can be calculated using the sensitivity and specificity of the test (i.e., the probability of a true positive plus the probability of a false positive).

Bayes' Theorem (Example 2)

- Let's assume that the prevalence of the disease is 1%, the sensitivity of the test is 90%, and the specificity of the test is 95%.
- Then, we can calculate the probability of having the disease given a positive test result as follows:
    - P(A|B) = P(B|A) * P(A) / P(B)
    - P(A|B) = 0.9 * 0.01 / (0.9 * 0.01 + 0.05 * 0.99)
    - P(A|B) = 0.15 or 15%
- This means that if a patient tests positive for the disease, there is a 15% chance that they actually have the disease, given the prevalence of the disease and the accuracy of the test.

Bayes' Theorem (Example 2)

- The posterior probability is a conditional probability that reflects the updated belief in a hypothesis or parameter value after observing new data.
- In Bayesian inference, we start with an initial belief about the probability of a hypothesis or the value of a parameter, called the prior probability. After observing new data, we update our belief using Bayes' Theorem to calculate the posterior probability, which incorporates the information from the observed data.
- In the medical test example, the prior probability is the probability of having the disease before the test is taken, and the posterior probability is the probability of having the disease given a positive test result. The posterior probability reflects the updated belief in the probability of having the disease, based on the new information provided by the positive test result.
- By using Bayes' Theorem to update our belief in light of new data, we can make more informed decisions and predictions in a wide range of fields, from medicine to finance to machine learning.

Bayes' Theorem: Exercise

1. *A bag I contains 4 white and 6 black balls while another Bag II contains 4 white and 3 black balls. One ball is drawn at random from one of the bags, and it is found to be black. Find the probability that it was drawn from Bag I.*

2. *A human is known to speak the truth 2 out of 3 times. The human throws a die and reports that the number obtained is a four. Find the probability that the number obtained is actually a four.*

3. *A student attempts 2 exams and can only score grades A and B (at least one f the scores has to be an A). What is the probability of scoring 2As i.e. A for both exams?*

End

Die - Truth Question

Question:
A human is known to speak the truth 2 out of 3 times. The human throws a die and reports that the number obtained is a four. Find the probability that the number obtained is actually a four.

Solution Alt 1: Considering the die i.e. assuming its a none biased six sided die.

In this example of using Bayes' Theorem to update our belief in light of new evidence we first define the following events:

- **T**: the event that the human speaks the truth = 2/3
- **L**: the event that the human lies = 1/3
- **X**: the event that the human **reports**
- **D**: the event that the die show the same result as reported.

We want to calculate P(T|R), the probability that the human is telling the truth given that they reported a particular result. Using Bayes' Theorem, we have:

```
P(T|X) = P(X|T) * P(T) / P(X)
```

- P(X|T) is the probability of the human reporting a particular result given that they are telling the truth. = **1/6** (Because we assume the die is six sided and fair)
- P(T) is the prior probability that the human is telling the truth.
- P(X) is the total probability of reporting the particular result, which can be calculated using the law of total probability:

```
P(X) = P(X|T) * P(T) + P(X|L) * P(L)
P(X) = (1 /6  * 2 / 3) + (1 / 6 * 1 / 3)
P(X) = 1 / 6
```

- P(X|L) is the probability of the human reporting the particular result given that they are lying, and P(L) is the prior probability that the human is lying.

Applying Bayes Theorem / Theory

```
P(T|X) = P(X|T) * P(T) / P(X)
P(T|X) = (1/6 * 2/3) / 1/6
P(T|X) = 2/3 OR 0.6667
```

Solution Alt 2: Focusing on the Truth - Lie aspect and not worrying about what type of dice this human is working with.

In this alternative, we purely focus on wheter what the human says is truthfull and care less about what is being done. We also assume that they have to speak hence probability of speaking is equal to 1.

Our solution thus becomes 2/3.

- If the probability of the human speaking is 1, then it means the human is always speaking. In this case, the probability that the human is telling the truth is simply the same as the probability that the human is truthful, which is given as 2/3.
- This is because the probability of the human telling the truth is the same as the probability that they are truthful, since they always speak. So, in this case, the probability that the human is telling the truth is 2/3.
- Essentially there is no clear inidaction of any additonal information that may lead us to think the huma is going to change his/her lying / truthfullness probability. i.e. no new info to update

---

These two alternatives are a classic exxample that indicate **when** Bayes Theorem can be usefull.

**BONUS:

1. Law of probability

The Law of Total Probability is a fundamental concept in probability theory that states that the total probability of an event can be expressed as the sum of the conditional probabilities of that event given each of a set of mutually exclusive and exhaustive events.

2. Simple Implementation

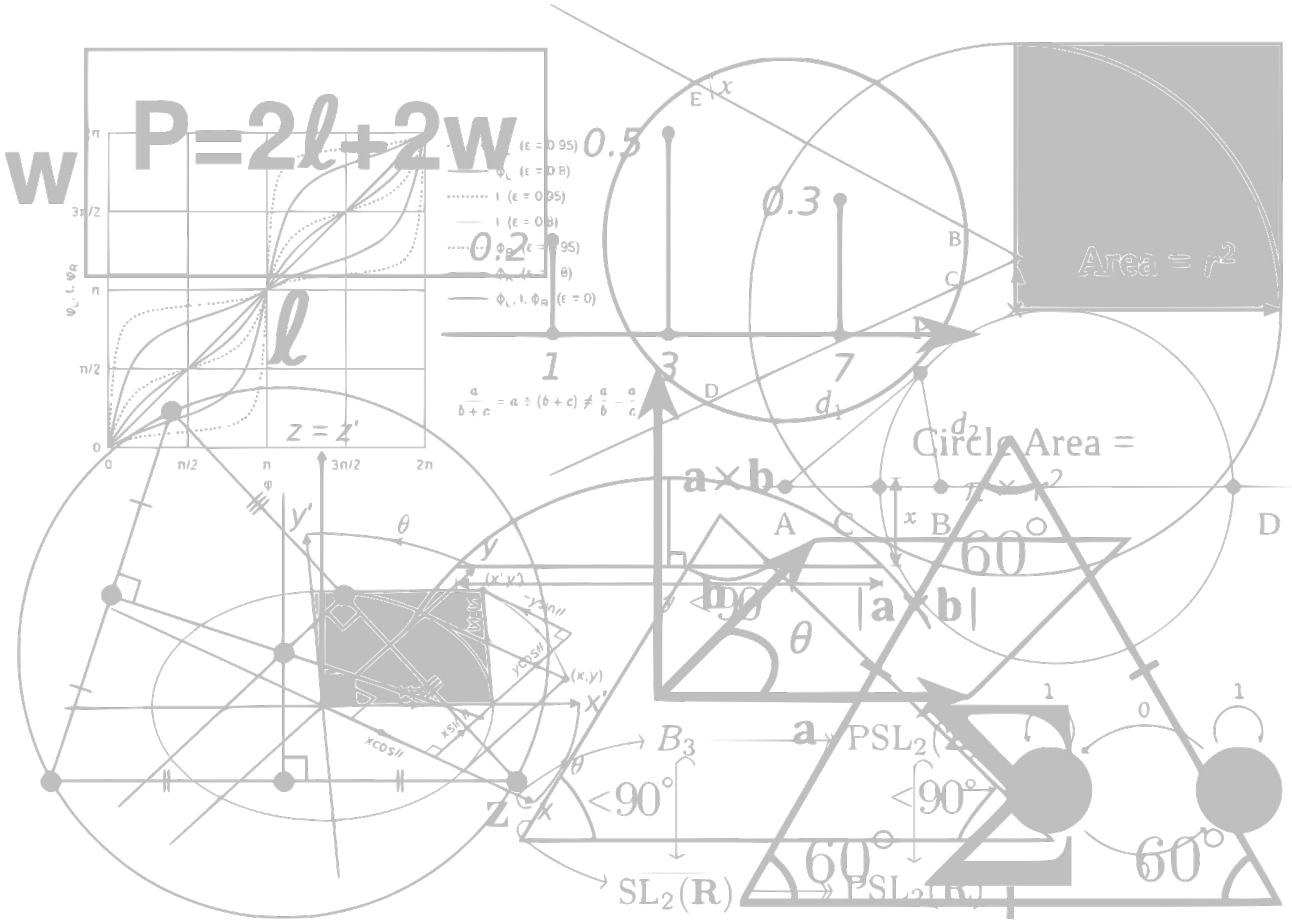
```
def  bayes_theorem(prior_prob, likelihood, evidence):
    posterior_prob = (prior_prob * likelihood) / evidence
    return posterior_prob

# Computes the posterior probability using Bayes' theorem.
# prior_prob: float, prior probability likelihood: float,
# likelihood of the evidence given the hypothesis
# evidence: float, probability of the evidence
```

Suppose we know that Cerebral Malaria affects 1% of the population. A test for the disease has a false positive rate of 5% and a false negative rate of 2%. If a randomly selected person tests positive for the disease, what is the probability that they actually have Cerebral Malaria?

Solution

```
# Prior probability of having the disease
prior_prob = 0.01

# Likelihood of testing positive given the person has the disease
true_positive = 0.98

# Likelihood of testing positive given the person does not have the disease
false_positive = 0.05

# Probability of testing positive
evidence = prior_prob * true_positive + (1 - prior_prob) * false_positive

# Calculate the posterior probability
posterior_prob = bayes_theorem(prior_prob, true_positive, evidence)

print("The probability that a person who tested positive actually has the d

The probability that a person who tested positive actually has the disease
```

Computational Learning Theory *(CLT)*



Computational Learning Theory

- Computational Learning Theory (CLT) is *a branch of statistics/machine learning/artificial intelligence (in general) which deals with fundamental bounds and theorems about analysing our (man and machine) ability to learn rules and patterns from data.*
- CLT provides mathematical models for understanding how people and machines learn and develop cognitive skills, such as decision-making strategies, problem-solving techniques, perception, language processing, and other forms of intelligence.
- It also studies the design of efficient learning algorithms, their computational complexity, and their performance guarantees under various conditions.
- There are many subfields of study, although perhaps two of the most widely discussed areas of study from computational learning theory are:

    i. *PAC Learning.*

    ii. *VC Dimension.*

Computational Learning Theory

What Questions Can Computational Learning Theory Answer?

i. How can you tell if an ML algorithm accurately represents your objective?

ii. How can you tell if the ML model successfully provided you with the correct results for a specific or general task?

iii. How many hypotheses should the machine come up with?

iv. What can you do to avoid overfitting?

v. How many examples of data are required?

Computational Learning Theory

What Are the Practical Applications of Computational Learning Theory?

- It can help programmers predict how well their algorithms will do in processing and analysing specific volumes of data. Will they work as well on 5 million data points as they did on 1 million data points? Were the results just as accurate?

- It simplifies the process of modifying algorithms by limiting the possible parameters, making software development and upgrading faster. Here, unsupervised learning plays a huge part, specifically in labelling data and choosing data points that are likely to produce the best results.

Hypothesis

- Suppose you want to build a model to predict whether a given email is spam or not based on its content. You have a dataset of labelled emails, where each email is labelled as spam or not spam.
- In this context, a *hypothesis would be a proposed relationship between the content of an email and its label (spam or not spam).* For example, a hypothesis could be that *emails containing certain keywords such as "earn money fast" or "free trial" are more likely to be spam.*
- To test this hypothesis, you would use a machine learning algorithm to train a model on your labeled dataset, using features such as the presence or absence of certain keywords to predict the label of each email. The trained model represents your hypothesis about the relationship between email content and spam/non-spam labels.
- You would then evaluate the performance of the model on a separate dataset of labeled emails that it has not seen before, to determine how well it generalizes to new data. If the model achieves high accuracy on the test set, you can be more confident in the hypothesis that emails containing certain keywords are more likely to be spam.

PAC Learning

- PAC learning is based on the idea that a learning algorithm should be able to produce a hypothesis that is probably approximately correct with respect to the underlying distribution from which the training data is drawn.
- A hypothesis is considered to be probably approximately correct if it has a small error on the training data and generalizes well to unseen data.
- The goal of a PAC learning algorithm is to find a hypothesis that has a small generalization error with high probability. In other words, the algorithm should produce a hypothesis that is probably approximately correct.

PAC Learning

To formalize this concept, let's define some terms:

i. Concept: A function that maps an input to an output. In supervised learning, the goal is to learn a concept from a set of labelled examples.

ii. Hypothesis: A candidate function that approximates the true concept. The hypothesis is learned by the learning algorithm.

iii. Sample: A set of labeled examples drawn from the underlying distribution.

iv. Error: The difference between the output of the hypothesis and the true label of an example.

v. Training error: The average error of the hypothesis on the training data.

vi. Generalization error: The expected error of the hypothesis on new, unseen data.

VC Theory

- VC *(Vapnik–Chervonenkis theory)* theory is a more sophisticated framework that provides a theoretical analysis of the generalization performance of a learning algorithm based on the capacity of the hypothesis class.
- The capacity of a hypothesis class measures its ability to fit different functions. The VC dimension is a measure of the capacity of a hypothesis class.
- It is defined as the maximum number of points that can be shattered by the hypothesis class. In other words, it measures the complexity of the hypothesis class.
- The VC theory provides bounds on the generalization error of a learning algorithm based on the VC dimension of the hypothesis class and the number of training examples.
- These bounds provide a way to assess the generalization performance of a learning algorithm without knowing the underlying distribution.
- In practice, the VC theory is often used to select the best hypothesis class for a given learning problem. The goal is to choose a hypothesis class that is powerful enough to fit the data but not too complex to overfit.
- The VC theory provides a way to balance the bias-variance tradeoff by considering the complexity of the hypothesis class.

Computational Learning Theory

Differences between PAC Learning and VC Theory:

- PAC learning is concerned with the learnability of concepts from examples, while VC theory is concerned with the generalization performance of a learning algorithm.
- PAC learning assumes that the training data is drawn from an underlying distribution, while VC theory does not make this assumption.
- PAC learning focuses on the accuracy of the hypothesis on the training data and generalization performance, while VC theory focuses on the capacity of the hypothesis class and the number of training examples.
- PAC learning is a simpler framework than VC theory and is easier to apply in practice. VC theory is a more sophisticated framework that requires more mathematical background and is more difficult to apply.

Computational Learning Theory

- *In summary, PAC learning and VC theory are two frameworks in machine learning that provide a theoretical analysis of the generalization performance of learning algorithms.*

- *While PAC learning is a simpler framework that focuses on the accuracy of the hypothesis on the training data and generalization performance, VC theory is a more sophisticated framework that provides bounds on the generalization error of a learning algorithm based on the capacity of the hypothesis class and the number of training examples.*

Linear Regression

Linear Regression

- *Linear regression is one of the easiest to implement machine learning algorithms, and can be seen a method used to define a relationship between a dependent variable (Y) and independent variable (X).*

$$y = mx + b$$

- Where y is the dependent variable, m is the scale factor or coefficient, b being the bias coefficient and x is the independent variable.
- The bias coefficient gives an extra degree of freedom to this model. The goal is to draw the line of best fit between x and y which estimates the relationship between x and y.
- We can calculate these these coefficient using different approaches.
    i. One is the Ordinary Least Mean Square Method approach and
    ii. The Gradient Descent approach.

Linear Regression: *Coefficients*

- The slope m of the regression line can be calculated using the following formula:

$$m = (sum((x - x\_mean) * (y - y\_mean))) / (sum((x - x\_mean) ** 2))$$

- To break down this formula, the numerator calculates the sum of the product of the deviation of each x value from its mean (x - x_mean) and the deviation of the corresponding y value from its mean (y - y_mean). The denominator calculates the sum of the squared deviation of each x value from its mean ((x - x_mean) ** 2). This gives us the variance of x

---

- Therefore, the slope m represents the covariance between x and y divided by the variance of x. It tells us how much y changes for every unit change in x.
- The y-intercept c of the regression line can be calculated using the following formula:

$$c = y\_mean - m * x\_mean$$

- This formula represents the point where the regression line intercepts the y-axis. It tells us the value of y when x is 0.

Linear Regression

General steps on how to perform linear regression for absolute beginners:

1. Collect Data: The first step is to collect data on the dependent and independent variables. For example, if you want to predict the price of a house, you might collect data on the size of the house, the number of bedrooms, the location, and the sale price.

2. Plot the Data: Plot the independent variable(s) on the x-axis and the dependent variable on the y-axis. This will give you a visual representation of the data and help you identify any patterns or trends.

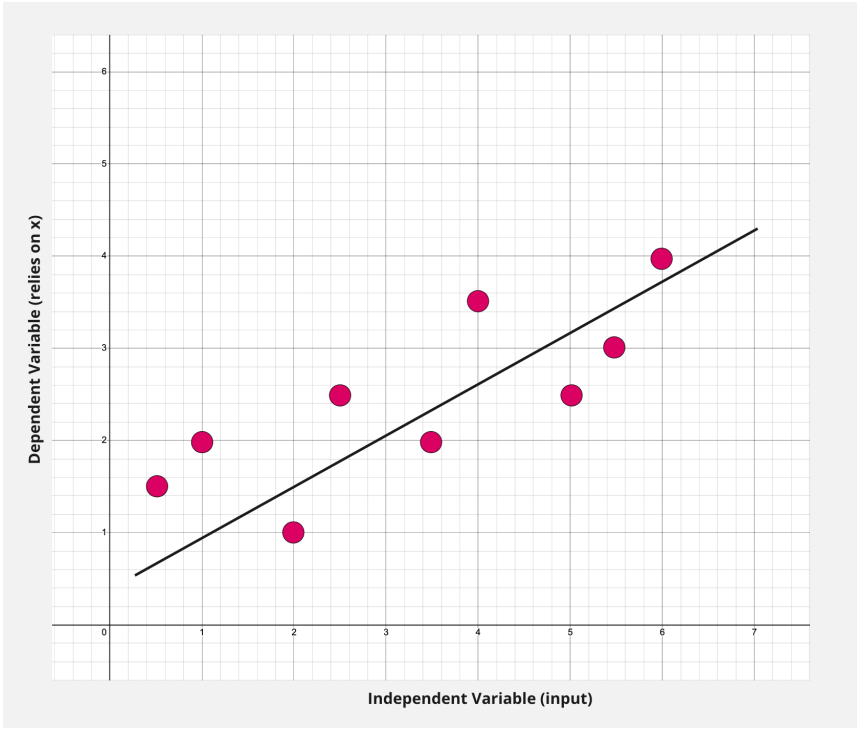
3. Calculate the Correlation Coefficient: The correlation coefficient measures the strength and direction of the relationship between the independent and dependent variables. You can use a statistical software or a spreadsheet to calculate the correlation coefficient.

4. Fit a Line: A linear regression model involves fitting a line to the data. The line should be the best fit for the data, meaning it should minimize the distance between the predicted values and the actual values. You can use a software or spreadsheet to find the line of best fit.

5. Interpret the Results: Once you have the line of best fit, you can use it to predict the value of the dependent variable for any given value of the independent variable. You can also calculate the coefficient of determination (R-squared), which measures how much o of the variation in the dependent variable can be explained by the independent variable.

Linear Regression

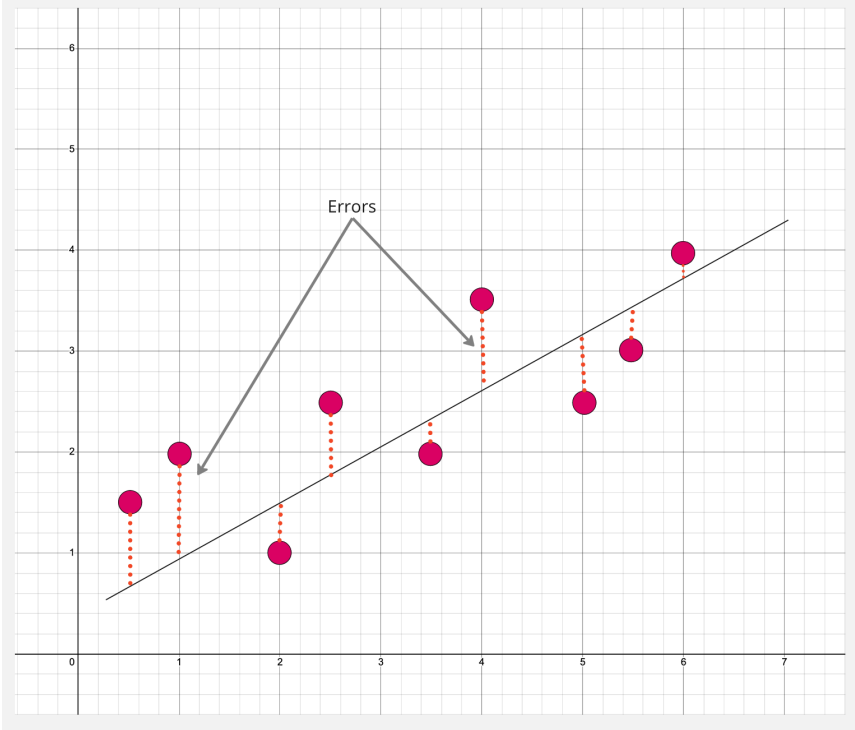
- If we had a sample inputs (x) and outputs (y) and we plot these scatter point on a 2d graph, we something similar to the graph below



- The line seen in the graph is the actual relationship we going to accomplish, And we want to minimize the error of our model. We'll try and achive this without overfitting and underfitting (We'll discuss these tow terms later in the subtopic.)

Linear Regression

- If we look closely, we notice that this line doesnt attempt to go through all points but tries to be close. This line is the best fit that passes through most of the scatter points and also reduces error which is the distance from the point to the line itself as illustrated below.



- The total error of the linear model is the sum of the error of each point, given by: $\sum_{i=1}^{n} r_i^2$
- Where, r_i = Distance between the line and i^{th} point and n =Total number of points.
- We are squaring each of the distance's because some points would be *above the line and some below*. We can minimize the error of our linear model by minimizing r

Linear Regression

- The previous illustration only shows how Linear Regression works for a 1 input 1 output scenario. We can however make modifications to accommodate other scenarios namely a multi-input – 1 output case.

- Given $y = w_1x_1 + b$, where
    - y is the predicted label (a desired output).
    - b is the bias (the y-intercept)
    - w1 is the weight of feature 1. Weight is the same concept as the "slope" *m* in the traditional equation of a line.
    - x1 is a feature (a known input).

- We simply include our new inputs x_1 ---- x_n and also incorporate all weights w_1---w_n as:

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + \ldots\ldots\ldots w_nx_n + b$$

Linear Regression *with Gradient Descent*

- Gradient descent is one of the most famous techniques in machine learning and used for training all sorts of neural networks. By definition it is an optimization algorithm used in machine learning to minimize the cost or loss function of a model.
- The basic idea behind gradient descent is to iteratively adjust the model parameters in the direction of steepest descent of the cost function, until a minimum is reached. At each iteration, the algorithm computes the gradient of the cost function with respect to the model parameters, which indicates the direction of maximum increase of the cost function.
- It then updates the parameters by subtracting a fraction of the gradient multiplied by a learning rate, which determines the step size of the update.
- Gradient descent can not only be used to train neural networks, but many more machine learning models. In particular, gradient descent can be used to train a linear regression model.

Linear Regression *with Gradient Descent*

- In the context of linear regression, gradient descent is an optimization algorithm used to find the values of the slope m and the intercept c that minimize the mean squared error (MSE) between the predicted values and the actual values of the dependent variable.
- Remember $y = mx + c$, in the previous approach, we used the mean of x and y to obtain the slope m then used this to calculate the y intercept thus resulting in our final equation. For GD we'll do this a bit differently by starting with some values then iterating over others until we have a reasonable MSE.
- The MSE is a cost function that measures the average squared difference between the predicted values y_pred and the actual values y of the dependent variable, over a given set of input features X. The formula for the MSE is:

$$MSE = \frac{1}{n} * sum((y_{pred} - y)^2)$$

- Where n is the number of data points in the dataset, and sum is the sum over all data points.
- The goal of gradient descent in linear regression is to find the values of m and c that minimize the MSE. To achieve this, the algorithm iteratively updates the values of m and c using the gradients of the cost function with respect to m and c. The gradient of the cost function with respect to m is: (where x is the input feature.)

$$\frac{dj}{dm} = \frac{2}{n} * sum((y\_pred - y) * x)$$

Linear Regression *with Gradient Descent*

- Similarly, the gradient of the cost function with respect to c is:

$$\frac{dj}{dc} = \frac{2}{n} * sum(y_{pred} - y)$$

- To update the values of m and c, the algorithm computes the gradients using the current values of m and c, and subtracts a fraction of the gradients multiplied by a learning rate alpha.

- The update rule for m is:

$$m = m - alpha * \frac{dj}{dm}$$

- And the update rule for c is:

$$c = c - alpha * \frac{dj}{dc}$$

- The learning rate alpha is a hyperparameter that controls the step size of the update. A high value of alpha may cause the algorithm to overshoot the minimum, while a low value of alpha may cause the algorithm to converge too slowly. Essentially this is what most call *learning_rate*

- The algorithm iteratively updates the values of m and c using the above update rules until the cost function converges or a maximum number of iterations is reached.

Linear Regression *with Gradient Descent*

- Given data points:

  x = [.5, 1, 2, 2.5, 3.5, 4, 5, 5.5, 6]) and,

  y = [1.5, 2, 1, 2.5, 2, 3.5, 2.5, 3, 4]

  i. Initialize the parameters m and c to random values.

  ii. Define a learning rate alpha and a number of iterations num_iters.

  iii. For each iteration, calculate the predicted values y_pred using the current values of m and c.

  iv. Calculate the cost function J, which measures the difference between the predicted values and the actual values of y.

  v. Update the values of m and c using the gradient of the cost function with respect to m and c.

  vi. Repeat steps 3-5 for num_iters iterations or until convergence.

A complete version of this question/sample is included in the Python Notebooks for this topic.

Refer to the relevant folder with shared Google Drive folder for the class.

End

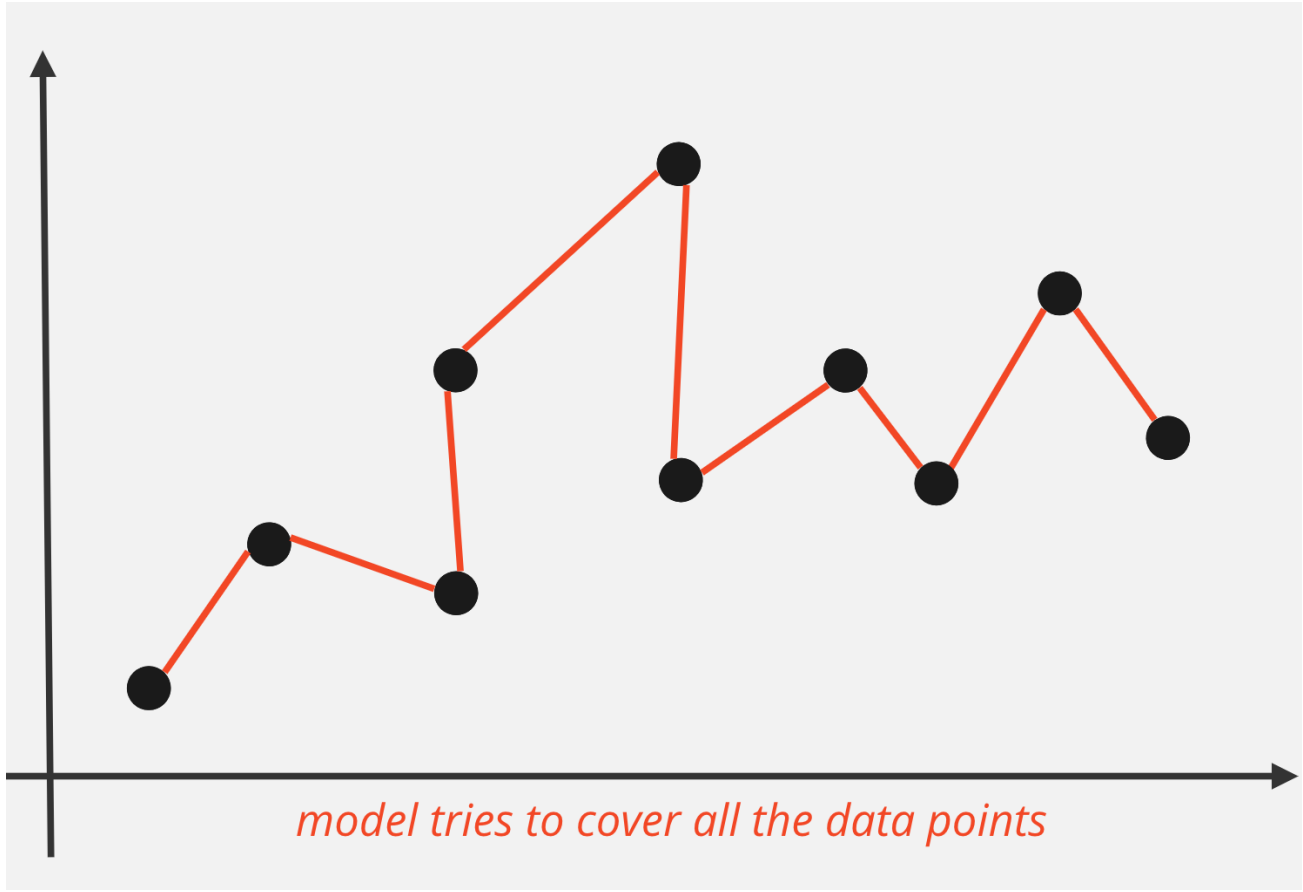
Generalization

Introduction

- Generalization refers to the ability of a model to perform well on new, unseen data that it has not been trained on.
- The goal of machine learning is not to simply memorize the training data, but to learn the underlying patterns and relationships that are present in the data, and to apply them to new, unseen data. This is what allows a machine learning model to be useful in real-world scenarios.
- Generalization is important because it ensures that the model is able to perform well on data that it has not seen before, which is critical in real-world applications where the data is constantly changing and evolving.
- To achieve good generalization, one needs to carefully balance between overfitting and underfitting.
    - Overfitting occurs when a model is too complex and fits the training data too closely,
    - Underfitting occurs when a model is too simple and is not able to capture the underlying patterns in the data.
- Other techniques for achieving good generalization include data augmentation, which artificially increases the size and diversity of the training data, and cross-validation, which is a technique for evaluating the performance of a model on multiple subsets of the data.
- Overall, achieving good generalization is critical for building machine learning models that can be deployed in real-world scenarios and provide accurate predictions on new, unseen data.

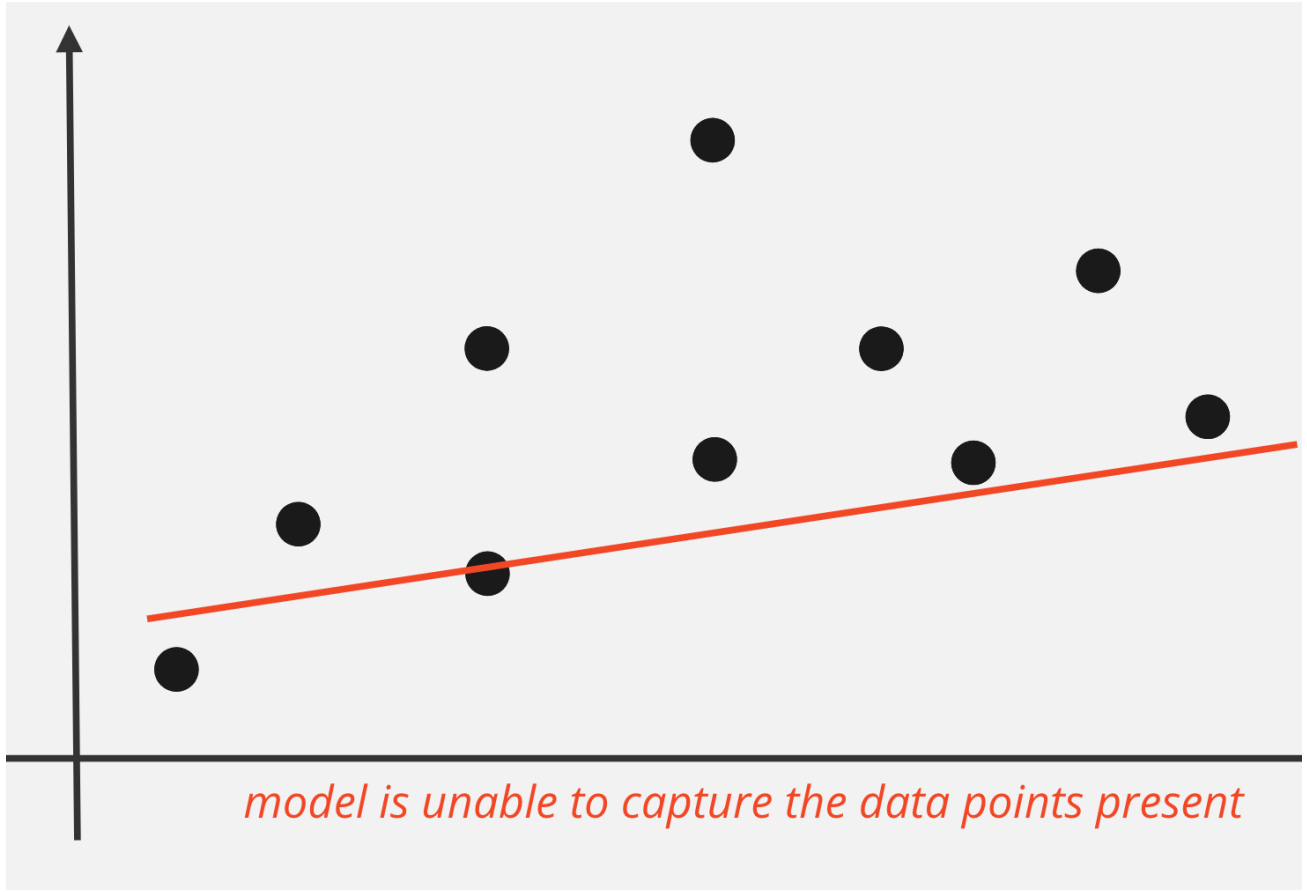
Overfitting

- Overfitting occurs when a model is too complex or flexible, and it fits the training data too closely, capturing the noise and random variations in the data.
- This can cause the model to perform poorly on new, unseen data, as it has essentially memorized the training data and is not able to generalize well to new data.



Underfitting

- Underfitting occurs when a model is too simple or inflexible, and it is not able to capture the underlying patterns and relationships in the data.
- This can also cause the model to perform poorly on new, unseen data, as it is not able to learn from the training data and generalize well to new data.



Possible Resolutions

- Both overfitting and underfitting cause the degraded performance of the machine learning model.
- To address overfitting, one can use techniques such as
    i. *regularization, which penalizes the model for being too complex, or*
    ii. *early stopping, which stops the training process before the model starts to overfit.*
- To address underfitting, one can use techniques such as
    i. *increasing the complexity of the model, adding more features or*
    ii. *transforming the data, or*
    iii. *using a different type of model altogether.*

Regularization

- This is a technique used to prevent overfitting by adding a penalty term to the cost function that the model is trying to optimize.
- A cost function is a mathematical function used to measure the error or loss between the predicted output of a model and the actual output.
- The goal of a machine learning model is to minimize the cost function during training by adjusting the model's parameters or weights. The cost function quantifies how well the model is performing on the training data, and provides a signal for the model to learn from.
- The basic idea behind regularization is to add a constraint on the model's parameters to prevent them from taking on extreme values, which can lead to overfitting. The penalty term is typically a function of the magnitude of the parameters, and is added to the cost function to create a new objective function that the model tries to minimize.
- There are two commonly used forms of regularization:
    - L1 regularization also know as: Lasso regularization
    - L2 regularization also know as: Ridge regularization

Regularization Illustration

- Suppose you have a bookshelf with different books on it. You want to choose the best book from the shelf that can help you with some task. However, some of the books might be too complex or too simple for the task, and you need to find the one that is just right.
- Now, let's imagine that each book on the shelf has a certain score that indicates how useful it is for the task. You want to choose the book with the highest score, but you also want to make sure that the book is not too complex or too simple for the task. This is where L1 and L2 regularization come in.
- *L1 regularization is like a bookshelf with a certain number of books*. *Some of the books might be too complex, while others might be too simple*, so you want to pick the book that is just right. To do this, *you remove the books with the lowest scores and keep only the books with the highest scores*. This way, you can choose the best book without getting distracted by the books that are too complex or too simple.
- *L2 regularization is like a bookshelf with different weights assigned to each book. Some books might be very heavy, while others might be very light.* You want to pick the book that is just right, but you also want to make sure that the book is not too heavy or too light. To do this, *you take the average weight of all the books and try to pick the book that is closest to this average weight*. This way, you can choose the best book without getting distracted by the books that are too heavy or too light.
- In summary, *L1 regularization focuses on keeping only the most important features (books)* and discarding the less important ones, while *L2 regularization focuses on shrinking the weights (sizes) of all features towards a middle ground*. Both techniques help prevent overfitting and improve the performance of the model on new, unseen data.

Regularization: *Ridge regression*

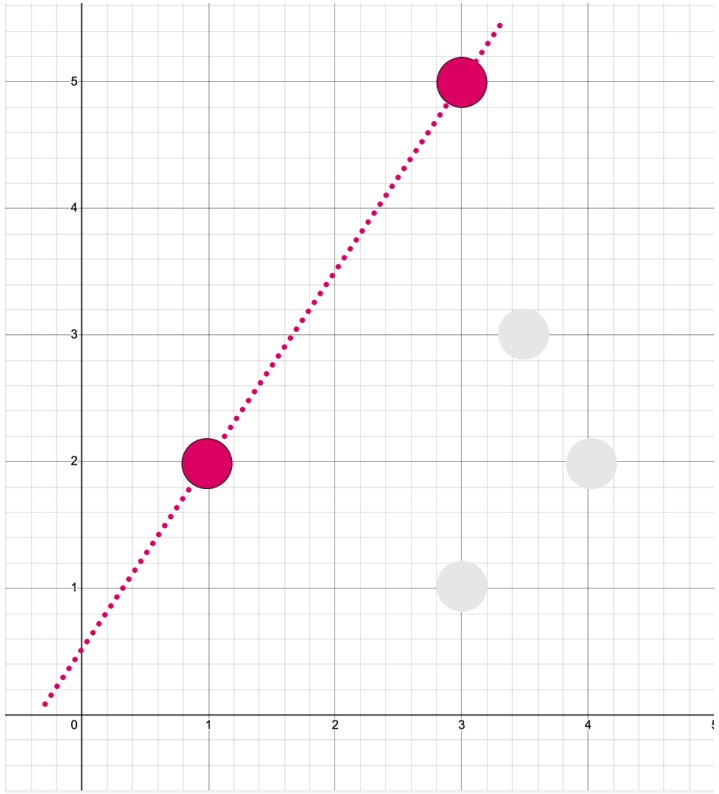
- This *adds a penalty term that is proportional to the square of the parameters*. This has the effect of shrinking all of the parameters towards zero, which can improve the generalization performance of the model by reducing the sensitivity to small changes in the input data.
- A small amount of bias is introduced so that we can get better long-term predictions. The amount of bias added to the model is called Ridge Regression penalty. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.

Say we have the points (1,2) and (3,5), our linear regression eq. would be:

- Step 1: Find the slope (m) The slope of the line can be calculated using the formula:
  - m = (y2 - y1) / (x2 - x1), Using the given points (1,2) and (3,5), we get: *m = (5 - 2) / (3 - 1), m = 3 / 2, m = 1.5*
  - Therefore, the slope of the line is 1.5
- Step 2: Find the y-intercept (c) The y-intercept is the value of y when x = 0. We can use one of the given points to find the y-intercept.
  - Using the point (1,2), we can substitute the values of x and y into the equation y = mx + c to solve for c: 2 = 1.5(1) + c, c = 2 - 1.5, c = 0.5
  - Therefore, the y-intercept of the line is 0.5

Regularization: *Ridge regression, cont.*

From the previous page, we'd have a plot as the one show below. This however means there is zero error between the line and the points hence our residual error and to be more specific our squared residual error are both Zero i.e. $(y\_pred - y)^2 = 0$



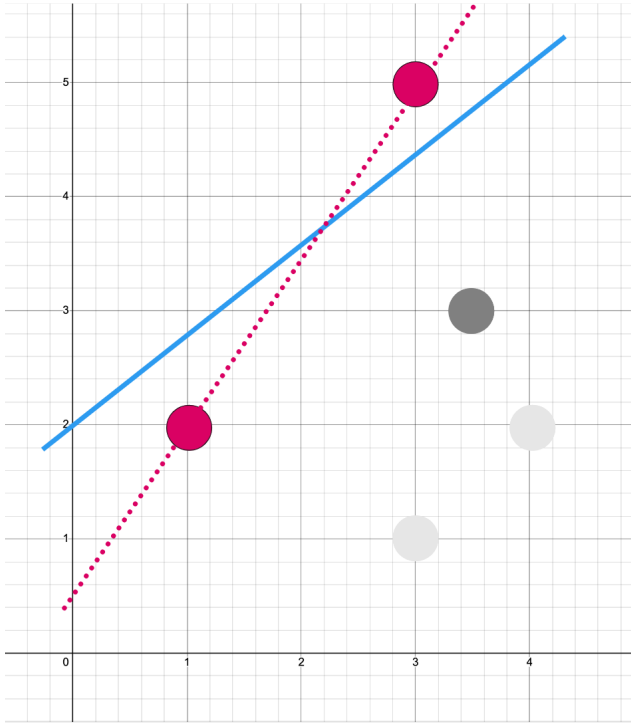
- Think of Ridge regression as a means to try and reduce the results of the following computation.

$$Sum\ of\ the\ squared\ residuals + Lambda * Slope^2$$

- Given that we currently have an over fit line:
- Sum of the squared residuals = *0* and thus we have: $0 + 1 * 1.5^2$ *(For now we'll use lambda as 1)*

Regularization: *Ridge regression*, *cont.*

Now, let us introduce another linear regression eq. (line) that doesn't overfit. This also has a much less slope i.e. $m = 0.8$. This is obtained from the new points of y(s) for the original two point i.e. from *1,2* and *3,5* we now have *1,2.8* and *3,4.4*



- We can start with the error sum of squared erros which will be given by:

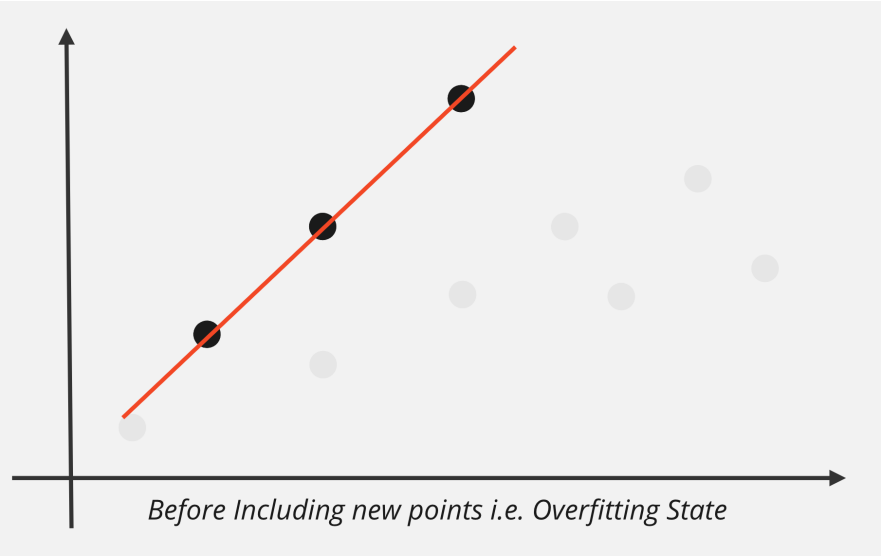
$$|2-2.8|^2 + |5-4.4|^2 = 0.64 + 0.36 = 1$$

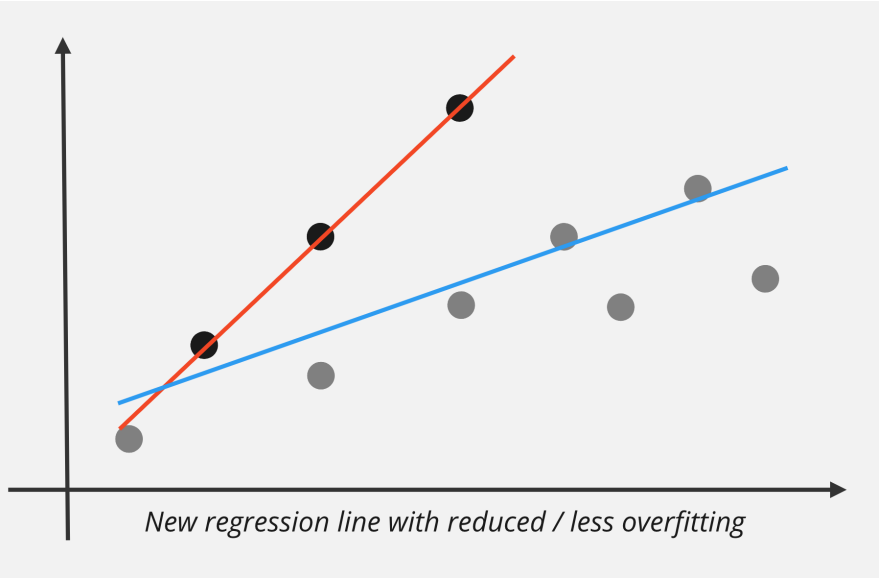
- If we plug these new values into our computation/forulae we have:

$$1 + 1 * 0.8^2 = 1.64$$

- Compared to the overfit result (2.25), 1.64 is clealry a smaller value hence this new line would be better off fo making feature computation of *y* for a given input *x*.
- *We can conclude that, the original line (Least squares fit) has a large varriance when we introduce more points while the ridge regression fit with the small amount of bias has a smaller varriance for the same case.*

Regularization: *Ridge regression*





- In these two illustrations, we see the danger of overfitting to the original points (items in black, think of these are training samples). Other samples (faint grey), are far from the regression line. This is a common scenario when one overfits a model to training samples gets a good performance then the introduces new samples and the model drastically reduces in performance.

- On the right, we have introduced a blue line, which represents a liner regression that avoids overfitting but ensures better future performance when new datapoints are seen.

Regularization: *Lasso Regression*

- Lasso regression is a type of linear regression that performs variable selection and regularization simultaneously. Lasso regression helps to address *Overfitting* by adding a penalty term to the objective function that encourages simpler models.
- The penalty term in lasso regression is based on the absolute value of the coefficients. This means that as the coefficients get larger, the penalty increases, which encourages some coefficients to shrink to zero. This has the effect of performing variable selection, since any variables with coefficients that are shrunk to zero are effectively removed from the model.
- The strength of the penalty in lasso regression is controlled by a hyperparameter called the regularization parameter, denoted by lambda (λ). A larger value of λ results in a stronger penalty, which in turn leads to more coefficients being shrunk to zero. A smaller value of λ results in a weaker penalty, which allows more coefficients to be non-zero.
- Lasso regression can be used for both prediction and inference. In the case of prediction, the goal is to use the model to make accurate predictions on new data. In the case of inference, the goal is to understand the relationships between the predictor variables and the response variable.
- Lasso regression is particularly useful when there are a large number of predictor variables and we want to identify the most important ones. By performing variable selection, lasso regression can help us identify the variables that are most strongly associated with the response variable, while ignoring the ones that are less important.

Ridge & Lasso

- Ridge Regression: *min(Sum of squared residiuals + lambda * $Slope^2$)*

- Lasso Regression: *min(Sum of squared residiuals + lambda * | Slope|)*

Why Lasso is usefull in feature selection:

Ridge regression and Lasso regression both recuce the slope however Ridge may get get close to zero but not actually equal to zero. This can be attributed to the precens of the squaring term. As a result Lasso regression can essentialy help us identify less impactfull features that may not be of any addtion when creating a model.

Ridge & Lasso regression differences

	L1 Regularization (Lasso)	*L2 Regularization (Ridge)*
1	Panelizes the sum of absolute value of weights.	penalizes the sum of square weights.
2	It has a sparse solution.	It has a non-sparse solution.
3	It gives multiple solutions.	It has only one solution.
4	Constructed in feature selection.	No feature selection.
5	Robust to outliers.	Not robust to outliers.
6	It generates simple and interpretable models.	It gives more accurate predictions when the output variable is the function of whole input variables.
7	Unable to learn complex data patterns.	Able to learn complex data patterns.
8	Computationally inefficient over non-sparse conditions.	Computationally efficient because of having analytical solutions

Feature Selection

Fs. Introduction

- Feature selection is a crucial step in machine learning that involves selecting a subset of relevant features from the available set of variables or attributes.
- By selecting the most informative features, we can improve the model's performance, reduce overfitting, and enhance interpretability. Here are some detailed notes on feature selection for machine learning:

Importance of Feature Selection:

1. Reduces dimensionality: Feature selection helps in eliminating irrelevant or redundant features, reducing the dimensionality of the dataset.
2. Improves model performance: By focusing on informative features, feature selection can enhance the model's predictive accuracy and generalization.
3. Reduces overfitting: Including irrelevant or redundant features can lead to overfitting, and feature selection mitigates this risk.
4. Enhances interpretability: Selecting meaningful features improves the interpretability of the model and enables better insights into the problem domain.

Fs. Types

- Filter methods: These methods rely on statistical measures to rank features based on their relevance to the target variable. Examples include correlation-based feature selection, mutual information, and chi-square tests.
- Wrapper methods: Wrapper methods evaluate feature subsets using a specific machine learning algorithm, measuring performance through cross-validation or a separate evaluation set. Examples include forward selection, backward elimination, and recursive feature elimination (RFE).
- Embedded methods: These methods incorporate feature selection as part of the model building process. They automatically select relevant features while training the model. Examples include LASSO (Least Absolute Shrinkage and Selection Operator) and tree-based feature importance.

Fs. Techniques

1. **Univariate Selection:** In this technique, features are selected based on their individual relationship with the target variable. Common statistical measures used are chi-square for categorical variables and ANOVA or correlation for numerical variables.

2. **Recursive Feature Elimination (RFE):** RFE is a wrapper method that recursively selects features by training the model iteratively. It starts with all features, ranks them based on importance, and eliminates the least important features until the desired number remains.

3. **Principal Component Analysis (PCA):** PCA is a dimensionality reduction technique that transforms the original features into a lower-dimensional space while preserving the most important information. The transformed components can be used as features.

4. **Regularization Methods:** Regularization techniques, such as L1 (Lasso) and L2 (Ridge) regularization, can be employed to penalize the coefficients of irrelevant features, effectively shrinking their impact and performing automatic feature selection.

5. **Tree-Based Methods:** Decision tree-based algorithms, such as Random Forests or Gradient Boosting Machines, provide feature importance scores based on how frequently a feature is used to make important decisions in the tree ensemble. These scores can guide feature selection.

PCA (This works for both FS and FE)

- Consider a dataset with three numerical features: X, Y, and Z. We want to apply PCA to reduce the dimensionality to two components.

  i. Compute the mean of each feature: mean_X, mean_Y, mean_Z.

  ii. Subtract the mean from each feature: X_centered = X - mean_X, Y_centered = Y - mean_Y, Z_centered = Z - mean_Z.

  iii. Construct the covariance matrix of the centered features: Cov_matrix = [[cov(X_centered, X_centered), cov(X_centered, Y_centered), cov(X_centered, Z_centered)], [cov(Y_centered, X_centered), cov(Y_centered, Y_centered), cov(Y_centered, Z_centered)], [cov(Z_centered, X_centered), cov(Z_centered, Y_centered), cov(Z_centered, Z_centered)]].

  iv. Compute the eigen-values and eigen-vectors of the covariance matrix.

  v. Sort the eigen-values in descending order and select the top two eigen-vectors.

  vi. Transform the original data into the new two-dimensional space: X_transformed = X_centered * eigenvector_1 + Y_centered * eigenvector_2.

RFE *(This is limited to Decision Trees)*

- Suppose we have a dataset with five features: A, B, C, D, and E. We want to perform RFE using a decision tree classifier and select the top two features.

    i. Start by training the decision tree classifier using all five features and evaluate its performance (e.g., accuracy, F1 score) on a validation set.

    ii. Rank the features based on their importance scores generated by the decision tree.

    iii. Remove the least important feature (e.g., E) and train the decision tree on the remaining features (A, B, C, D).

    iv. Evaluate the performance of the decision tree on the validation set again.

    v. Repeat the process iteratively, removing one feature at each step, until only two features remain (e.g., A and B).

    vi. Select the final two features (A, B) based on the best performance achieved by the decision tree during the elimination process.

Univariate

Given data with four features: P, Q, R, and S. We want to perform univariate feature selection using the chi-square test for a binary classification problem.

i. Split the dataset into a feature matrix X and a target vector y.

ii. Compute the chi-square statistic and p-value for each feature's relationship with the target variable.

iii. Sort the features based on their chi-square statistics or p-values.

iv. Choose the desired number of top features (e.g., two features) with the highest chi-square statistics or lowest p-values.

v. Select the final features (e.g., P, Q) based on the ranking obtained.

Fs. Evaluation

Performance metrics

The choice of performance metrics depends on the specific machine learning task. Common metrics include accuracy, precision, recall, F1 score, or area under the receiver operating characteristic curve (AUC-ROC).

Cross-validation

It is essential to evaluate feature selection techniques using cross-validation to ensure the robustness of the selected features. Cross-validation helps estimate the model's performance on unseen data and reduces the risk of overfitting.

Domain knowledge:

Incorporating domain knowledge is crucial when selecting features. Experts in the problem domain can provide insights into which features are likely to be relevant and should be considered during the feature selection process.

Feature Selection Best Practices

1. *Iterative process:* Feature selection should be an iterative process. Start with a larger set of potential features and gradually refine the selection based on evaluation results. It may involve trying different techniques, feature combinations, or thresholds.

2. *Feature interaction:* Consider interactions between features, as certain combinations of features may provide more predictive power than individual features alone. Feature engineering techniques like polynomial features or interaction terms can be helpful in capturing such interactions.

3. *Data leakage:* Be cautious of data leakage when performing feature selection. Data leakage occurs when information from the validation or test set is inadvertently used during feature selection. Ensure that feature selection is based only on training data and is independent of the target variable.

4. *Feature importance stability:* Assess the stability of feature importance rankings by repeating feature selection with different train-test splits or cross-validation folds. Features that consistently appear as important across multiple iterations are more reliable.

5. *Balance between simplicity and performance:* While it is desirable to have a concise set of features, overly aggressive feature selection may discard potentially useful information. Strike a balance between simplicity and model performance, considering the trade-off between interpretability and accuracy.

6. *Evaluate multiple algorithms:* Different algorithms may have different feature requirements. Evaluate feature selection techniques using various algorithms to ensure that the selected features generalize well across different models.

7. *Re-evaluate feature selection after model selection:* If the feature selection process is performed before model selection, it may be necessary to re-evaluate the selected features using the final chosen model. Different models may have different sensitivities to features, and their performance with the selected features should be assessed.

Feature Extraction

Fe. Introduction

Feature extraction is a crucial step in machine learning that involves transforming raw input data into a set of representative features that capture relevant information for a particular task. These features serve as inputs to machine learning algorithms, enabling them to learn patterns and make predictions or classifications. Here are some detailed notes on feature extraction for machine learning:

Importance of Feature Extraction:

- Feature extraction helps in reducing the dimensionality of the data, which can enhance the performance of machine learning algorithms and reduce computational complexity.
- It assists in focusing on the most relevant information, discarding irrelevant or redundant data, and improving the generalization ability of the model.
- Proper feature extraction can help uncover hidden patterns or structures in the data, making it easier for the algorithm to learn and make accurate predictions.

Fe. Types

- Handcrafted Features: These features are designed manually by domain experts based on their knowledge and understanding of the problem domain. Handcrafted features can be specific to the problem at hand and can provide good performance if chosen wisely.
- Automated Feature Extraction: This approach involves using algorithms or techniques to automatically extract features from raw data. Automated methods can be broadly classified into two categories:
- Transformation-based Methods: These methods apply mathematical transformations, such as Fourier transform, wavelet transform, or Principal Component Analysis (PCA), to extract features from the data.
- Feature Selection Methods: These methods aim to select the most informative subset of features from the original feature set. Examples include filter methods (e.g., correlation-based feature selection) and wrapper methods (e.g., recursive feature elimination).

FE. Techniques

- Statistical Features: Statistical measures like mean, variance, skewness, and kurtosis can provide insights into the distribution and central tendencies of the data.
- Frequency Domain Features: Transforming data into the frequency domain using techniques like Fourier or wavelet transforms can extract features related to the frequency components of the data.
- Spatial Domain Features: For image or spatial data, features like edges, textures, corners, or color histograms can capture important characteristics.
- Deep Learning-based Features: Convolutional Neural Networks (CNNs) can automatically learn hierarchical features from raw data, particularly for tasks like image recognition. Features can be extracted from intermediate layers of the network.
- Natural Language Processing (NLP) Features: For text data, techniques like word embeddings (e.g., Word2Vec or GloVe) or bag-of-words representations can be used to extract features.

Feature Extraction Best Practices

1. *Understand the Problem and Domain: Gain a solid understanding of the problem you are trying to solve and the domain in which it resides. This knowledge will guide your feature extraction process, allowing you to identify relevant characteristics and potential feature engineering techniques specific to the problem.*
2. *Pre-process the Data: Before performing feature extraction, pre-process the data to handle missing values, outliers, or other data quality issues.*
3. *Explore and Visualize the Data: Perform exploratory data analysis (EDA) to gain insights into the distribution, relationships, and patterns within the data.*
4. *Consider Feature Scaling: If you are using certain algorithms that are sensitive to the scale of the features (e.g., distance-based algorithms like k-means or support vector machines), it is advisable to apply feature scaling to ensure that all features contribute equally.*
5. *Utilize Domain Knowledge and Insights: Leverage your domain expertise to guide the feature extraction process.*
6. *Experiment with Different Techniques: Explore various feature extraction techniques such as statistical measures, frequency domain analysis, spatial domain analysis, deep learning-based approaches, or natural language processing (NLP) methods depending on the type of data you are working with.*
7. *Evaluate and Validate Extracted Features: Assess the quality and relevance of the extracted features by evaluating their performance within the machine learning pipeline.*
8. *Iterate and Refine: Feature extraction is an iterative process. Continuously evaluate and refine the feature extraction pipeline by incorporating feedback from model performance, domain knowledge, or new insights gained from further analysis.*

Feature Selection vs
Feature Extraction

Feature selection

- Aims to identify and select a subset of the original features that are most relevant to the target variable or prediction task.
- It focuses on retaining a subset of the original features while discarding the rest.
- Feature selection methods can be categorized into filter methods, wrapper methods, and embedded methods.
- Filter methods evaluate the relevance of features based on statistical measures or information theory and rank or score them accordingly.
- Wrapper methods use a machine learning algorithm to evaluate different subsets of features and select the subset that achieves the best performance.
- Embedded methods incorporate feature selection as part of the model training process, where feature importance is determined during the training of the algorithm.
- Feature selection can be computationally efficient and easier to interpret, as it retains the original features. However, it may overlook potential interactions or combinations of features that are important for the prediction task.
- Feature selection is useful when the original feature set is relatively small, and there is a clear understanding of the relevant features.

Feature extraction

- Involves transforming the original features into a new set of representative features.
- It aims to capture the essential information from the original features and create a reduced representation of the data.
- Feature extraction methods can be broadly categorized into transformation-based methods and deep learning-based methods.
- Transformation-based methods apply mathematical transformations to the data to extract new features.
- Deep learning-based methods, such as Convolutional Neural Networks (CNNs), learn hierarchical representations of the data through multiple layers and extract features from intermediate layers.
- Feature extraction can uncover hidden patterns or structures in the data that may not be evident in the original features.
- It can be effective when dealing with high-dimensional data or when the original feature set is noisy or redundant. However, feature extraction can result in a loss of interpretability, as the new features may not have a direct correspondence to the original features.
- Feature extraction requires more computational resources, especially when using deep learning-based methods, as they involve training complex models.

Model Selection

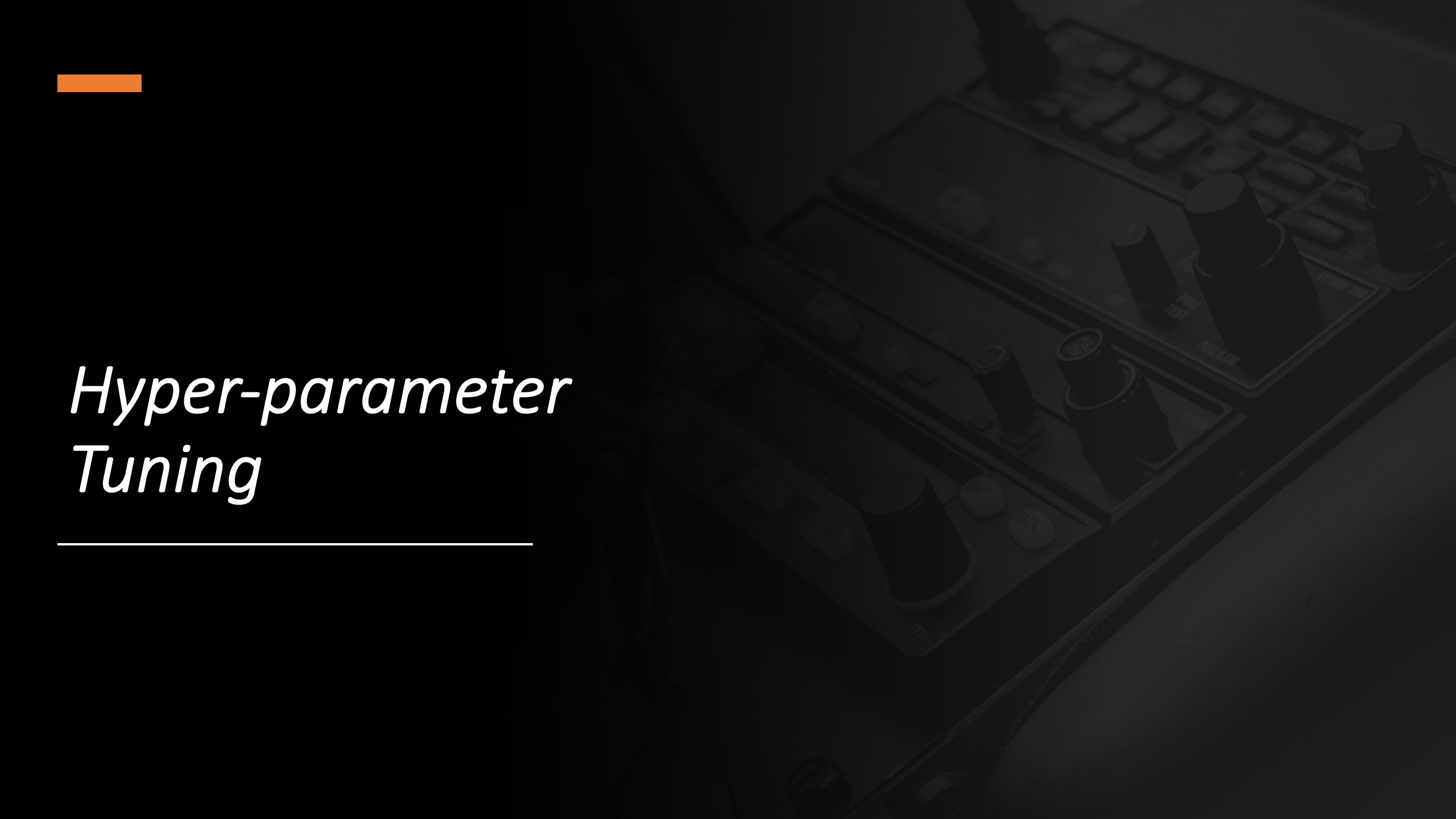
Model Selection Considerations

- **Define the problem:** Clearly understand the problem you are trying to solve and the objectives you want to achieve. This will help you identify the appropriate types of models to consider.
- **Identify candidate models:** Based on your problem definition, research and identify a range of candidate models that are commonly used for similar problems. Consider different types of models, such as linear regression, decision trees, support vector machines, neural networks, etc.
- **Specify evaluation criteria:** Determine the evaluation criteria that you will use to compare and select the best model. This can include metrics like accuracy, precision, recall, F1 score, mean squared error, area under the curve (AUC), etc. The choice of evaluation criteria depends on the nature of the problem and the goals of the project.
- **Split the data:** Divide your available data into training, validation, and test sets. The training set is used to train the models, the validation set is used for model selection, and the test set is used to evaluate the final selected model. The split can be, for example, 70% for training, 15% for validation, and 15% for testing.
- **Choose a performance measure:** Select a performance measure based on your evaluation criteria. For example, if you are interested in classification accuracy, you may choose cross-validation or accuracy on the validation set as the performance measure.
- **Implement candidate models:** Implement the candidate models using suitable machine learning libraries or frameworks. Train each model on the training set and evaluate their performance on the validation set using the chosen performance measure.

Model Selection Considerations

- **Compare performance:** Compare the performance of the candidate models based on the chosen performance measure. Consider the mean performance as well as the variance or stability of the models' performance. Visualize the results using plots or tables to aid in the comparison process.
- **Tune hyperparameters:** Identify the hyperparameters of the models that can be adjusted to optimize their performance. Perform hyperparameter tuning by trying different combinations of values for these parameters and selecting the set that gives the best performance on the validation set. Techniques like grid search, random search, or Bayesian optimization can be used for hyperparameter tuning.
- **Assess model complexity**: Consider the complexity of the models and the trade-off between model complexity and generalization performance. A simpler model may generalize better to unseen data, but it may have lower performance on the training set. Use techniques like cross-validation or learning curves to analyze the bias-variance trade-off.
- **Select the best model:** Based on the performance comparison, hyperparameter tuning, and assessment of model complexity, select the model that performs the best according to the chosen evaluation criteria. This model will be considered as the final selected model.
- **Evaluate the final model:** Evaluate the final selected model on the test set, which provides an unbiased estimate of its performance on unseen data. This evaluation gives a more reliable assessment of the model's generalization capability.
- **Iterate if necessary:** If the performance of the selected model is not satisfactory or if there are new insights or requirements, consider iterating the model selection process by trying different models, feature engineering techniques, or data preprocessing methods.

Hyper-parameter Tuning



Hyper-parameter

- Machine Learning models are composed of two different types of parameters:
    i. Hyperparameters = are all the parameters which can be arbitrarily set by the user before starting training (eg. number of estimators in Random Forest).
    ii. Model parameters = are instead learned during the model training (e.g. weights in Neural Networks, Linear Regression).
- Hyperparameters are the parameters that cannot be estimated by the model itself and you need you specify them manually. These parameters affects you model in many ways for a specific set of data.
- For you model/function to perform best you have choose the best set of hyperparameters. Hyperparameter optimization or hyperparameter tuning is one of the most Important part that everyone should consider.
- Examples are max depth in Decision Tree, penalty term in Ridge Regression and learning rate in deep neural network. These hyperparameters have a direct effect on whether your model will be Underfitting or overfitting.
- Remember that for different data set our optimal hyperparameters will change. There are two *main* ways to find the best hyperparameter setting .
    1. Manual search
    2. Automated search

Manual Search.

- When using Manual Search, we choose some model hyperparameters based on our judgment/experience. We then train the model, evaluate its accuracy and start the process again.
- This is hard and require experiment tracker. If you're researching on tuning then it's good as it gives you control over the process. But remember this is not practical method for hyperparameter search. With the huge amount of trials it is very expensive and time-consuming.
- Assume you are working with a Random Forest Classifier where the main parameters used by are:
    - *criterion = the function used to evaluate the quality of a split.*
    - *max_depth = maximum number of levels allowed in each tree.*
    - *max_features = maximum number of features considered when splitting a node.*
    - *min_samples_leaf = minimum number of samples which can be stored in a tree leaf.*
    - *min_samples_split = minimum number of samples necessary in a node to cause node splitting.*
    - *n_estimators = number of trees in the ensemble.*
- *we will have to specify the number of each and this may or may not lead to any improvement in perfomance.*

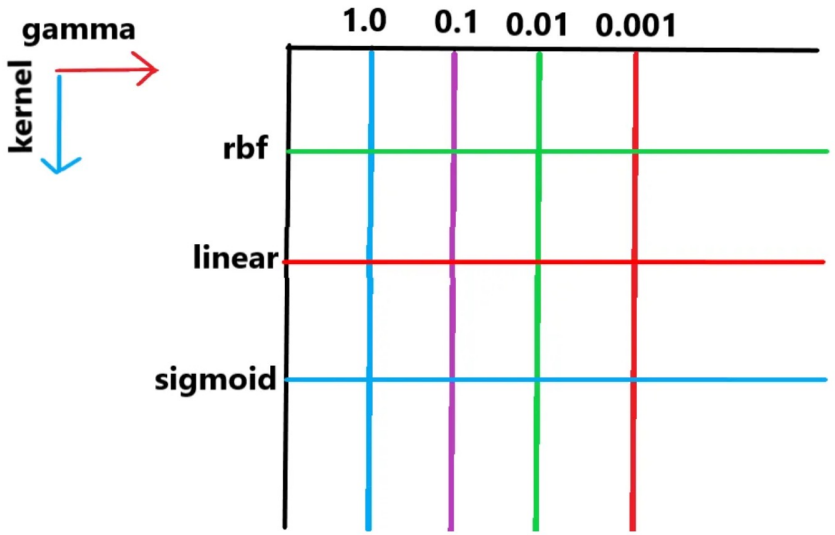
Automated Search (Random)

- As the name suggests we take the hyperparameter combination at random. That *means unlike Grid Search where we used every possible hyperparameter combination* here we sample values from a statistical distribution for each hyperparameter. A sampling distribution is defined for every hyperparameter to do a random search.
- Random search explores different areas of the search space, which can be advantageous when the impact of hyperparameters is not well understood or when the search space is large. Random search is particularly effective when only a few hyperparameters have a significant impact on model performance, as it allows for a more efficient exploration of the space.
- It may require fewer evaluations compared to grid search, especially when the search space is high-dimensional. Random search is less likely to get stuck in local optima since it explores various regions of the search space.
- However, there is a chance that random search might miss the most optimal hyperparameter combination.Randomized search let's you take control over the number of iteration to be computed. Although it may not be as accurate as Grid Search.

Automated Search (Grid Search)

- In Grid Search, we set up a grid of hyperparameters and train/test our model on each of the possible combinations. In order to choose the parameters to use in Grid Search, we can now look at which parameters worked best with Random Search and form a grid based on them to see if we can find a better combination.
- It considers all possible combinations of hyperparameters and covers the entire search space hence provides a systematic and comprehensive approach to hyperparameter tuning. Grid search is useful when the search space is small and the impact of hyperparameters is well understood.
- Grid search guarantees that the best hyperparameter combination within the defined search space will be found, given enough computational resources and evaluation time.
- However, it can be computationally expensive and time-consuming, especially when dealing with a large number of hyperparameters or a wide range of potential values.
- Lets say we have two:
    - ‘gamma’ : [1, 0.1, 0.01,0.001]
    - ‘kernel’ : [‘rbf’, ‘linear’, ‘sigmoid’]

Automated Search (Grid Search)



- We can see the grid formed here. Now what the algorithm does is for each of the inter section point it runs the model with the hyperparameter combination It have.
- Like in the first iteration the hyperparameter set will be [‘rbf’, 1.0] then in the next iteration [‘rbf’, 0.1] etc.
- After running all of the grid points it will choose the best performing model and return it. Remember the name Grid search CV , the CV stands for Cross Validation.

Comparison

1. Random search allows for a more efficient exploration of the search space, especially in high-dimensional spaces, while grid search covers the entire space but can be computationally expensive.

2. Random search is better suited when the impact of hyperparameters is not well known, and grid search is beneficial when there is a clear understanding of the impact of hyperparameters and a small search space.

3. Random search has a higher chance of missing the optimal hyperparameter combination, while grid search guarantees finding the best combination within the defined search space.

4. Random search can be more practical and effective in many scenarios, but grid search is still valuable for smaller search spaces or when exhaustive evaluation is feasible.

Other Techniques

1. **Bayesian Optimization**: Bayesian Optimization builds a probabilistic model of the objective function and iteratively selects the most promising hyperparameters to evaluate. It uses a combination of exploration and exploitation to efficiently search the hyperparameter space. Bayesian Optimization can handle both continuous and discrete hyperparameters.

2. **Genetic Algorithms:** Genetic Algorithms are inspired by the process of natural selection. They use a population-based approach, where a set of hyperparameter configurations (individuals) evolve over multiple generations. Individuals with higher fitness (better performance) have a higher chance of being selected for reproduction, creating new individuals with different combinations of hyperparameters.

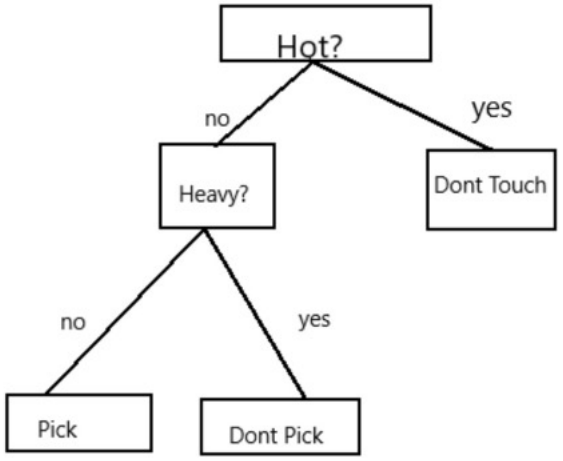
3. **Gradient-based Optimization:** Gradient-based optimization methods leverage gradient information to find the optimal hyperparameters. Techniques like Gradient Descent and its variants can be used to update the hyperparameters based on the gradient of the model performance with respect to the hyperparameters. However, this approach requires the objective function to be differentiable.

4. **Automated Machine Learning (AutoML) Libraries**: There are several AutoML libraries available that provide end-to-end automated solutions for hyperparameter tuning. These libraries often combine multiple techniques and provide a user-friendly interface for automated hyperparameter optimization. Examples include Auto-sklearn, TPOT, and H2O AutoML.

Decision Trees

Decision Trees

- One of the most popular machine learning algorithms out there, Decision Trees are by far the easiest to understand in how they function.
- Unlike Logistic Regression or Support Vector Machine which requires a solid mathematical foundation to be understood, Decision Trees mimic the way we Humans operate on a daily basis. for example: Let's say we have a pot and we would like an agent to pick it up but at the same time, we don't want to burn our hand or drop it because of the weight.

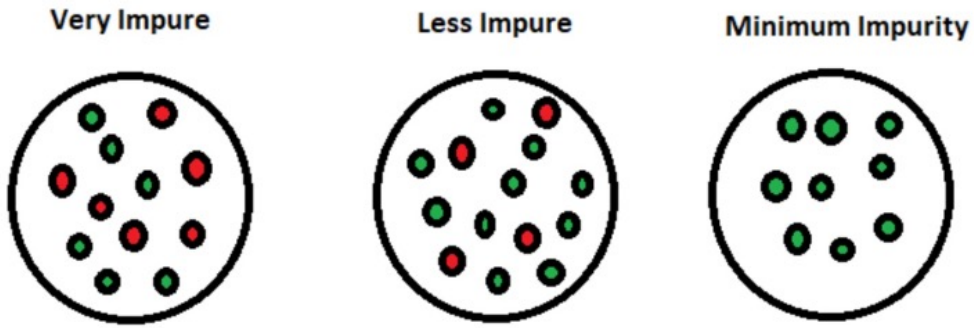


- *The first Box from where the split starts is known as the "Root Node".*
- *The Nodes in the middle are called Internal Nodes*
- *and the last two Nodes where the Tree ends are the "leaf" or Terminal Nodes.*

Decision Trees: Terms

Entropy: *Given by:* $E = -\sum_{i=1}^{N} p_i log_2 p_i$

- In machine learning, entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. *(measures the impurity or uncertainty in a group of observations):*



Information Gain: *Given by:* $Gain = E_{parent} - E_{children}$

- Information gain can be defined as the amount of information gained about a random variable or signal from observing another random variable. It can be considered as the difference between the entropy of parent node and weighted average entropy of child nodes.

Decision Trees: Entropy Formula

- Σ (P(x) * log2(P(x))) OR $E = -\sum_{i=1}^{N} p_i log_2 p_i$

where:

- Entropy represents the overall entropy of the dataset.
- Σ denotes the sum.
- P(x) represents the probability of a data point belonging to a particular class.

1. Count the occurrences of each class in the dataset.
2. Calculate the probability of each class by dividing the count of that class by the total number of data points.
3. For each class probability, calculate the product of the probability and the logarithm base 2 of that probability.
4. Sum up the products from step 3 for each class.
5. Multiply the sum from step 4 by -1 to obtain the entropy value.

- The entropy value ranges from 0 to log2(N), where N is the number of classes. A value of 0 indicates a completely pure dataset, where all data points belong to the same class. A higher entropy value indicates higher impurity and greater uncertainty in the dataset. The decision tree algorithm aims to minimize the entropy at each node by selecting the splits that result in the greatest reduction in entropy, leading to more homogeneous subsets and better classification.

Decision Trees: Algorithms

- **Random Forest:** Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. Each decision tree is trained on a different random subset of the training data, and the final prediction is obtained by aggregating the predictions of individual trees.
- **ID3 (Iterative Dichotomiser 3):** ID3 is one of the earliest decision tree algorithms. It uses information gain as the criterion to make decisions at each node. Information gain measures the reduction in entropy or impurity after splitting the data based on a particular attribute.
- **C4.5:** C4.5 is an extension of the ID3 algorithm. It improves upon ID3 by handling both categorical and continuous attributes. C4.5 uses information gain ratio instead of information gain to overcome the bias towards attributes with a large number of outcomes.
- **CART** (Classification and Regression Trees): CART is a widely used decision tree algorithm that can be used for both classification and regression tasks. It uses the Gini impurity or the mean squared error as the splitting criterion, depending on whether the task is classification or regression.
- **Gradient Boosting:** Gradient Boosting is another ensemble method that combines decision trees. It builds decision trees in a sequential manner, where each subsequent tree corrects the mistakes made by the previous trees. The final prediction is obtained by summing the predictions of all the trees.
- **Chi-Squared Automatic Interaction Detection (CHAID):** CHAID is a decision tree algorithm that is primarily used for categorical target variables. It uses the chi-squared test to determine the statistical significance of the relationships between predictor variables and the target variable.

ID3 (Iterative Dichotomiser 3) Example 1

- Given a dataset:

Feature 1	Feature 2	Label
a	*x*	0
b	*x*	0
c	*y*	1
c	*y*	1
b	*x*	2

- *Feature 1: 1/5 – a , 2/5 – b , 2/5 – c*
- *Feature 2: 3/5x, 2/5 y*
- *Labels: 2/5 – 0 , 2/5 – 1, 1/5 – 2*

- Since we have 2 features, we can use the concept of information gain to decide which becomes the root node question. For this we'll look at the entire dataset and calculate:
    i. *Class/Label Entropy*
    ii. *Feature 1 Entropy*
    iii. *Feature 2 Entropy*
- We can then use the resulting values to obtain Information Gain for both features and use the greater as to decide the first question to ask at the root node.

ID3 (Iterative Dichotomiser 3) Example 1

Entropy Calculations

Class / Label Entropy: $-\left(\frac{2}{5}\log_2\left(\frac{2}{5}\right)+\frac{2}{5}\log_2\left(\frac{2}{5}\right)+\frac{1}{5}\log_2\left(\frac{1}{5}\right)\right)$ **= 1.522**

Feature 1 Entropy:

$$\frac{1}{5}\left(-\frac{1}{1}\log_2\left(\frac{1}{1}\right)\right)+\frac{2}{5}\left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right)-\frac{1}{2}\log_2\left(\frac{1}{2}\right)\right)+\frac{2}{5}\left(-\frac{2}{2}\log_2\left(\frac{2}{2}\right)\right)$$

Feature 1: **a**, Label **0**

Feature 1: **a**, Label **0** & **2**

Feature 1: **a**, Label 1 *(2 instances)*

= 0.4

Feature 2 Entropy:

$$\frac{3}{5}\left(-\frac{2}{3}\log_2\left(\frac{2}{3}\right)-\frac{1}{3}\log_2\left(\frac{1}{3}\right)\right)+\frac{2}{5}\left(-\frac{2}{2}\log_2\left(\frac{2}{2}\right)\right)$$

Feature 2: x, Label **0** & **2**

Feature 2: **y**, Label **1**

= 0.551

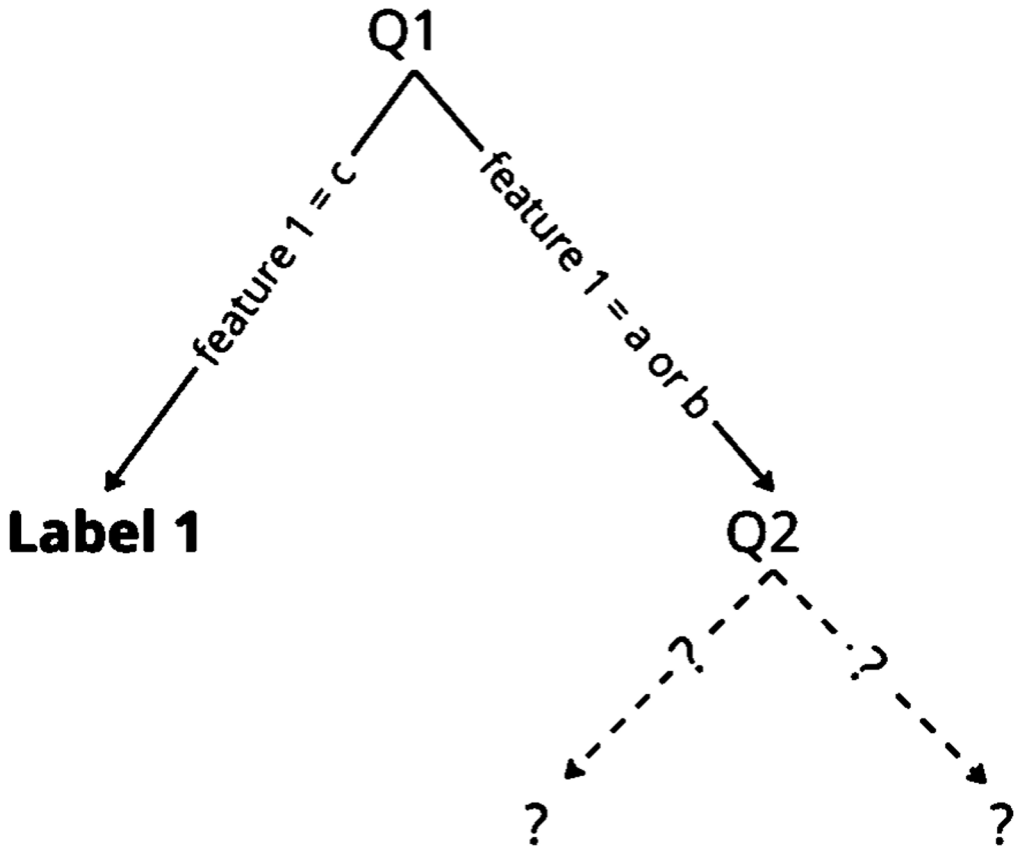
ID3 (Iterative Dichotomiser 3) Example 1

Information Gain (IG) Calculations:

(parent/class/label entropy – feature/child entropy)

- *IG for Feature 1 = Class E – Feature 1 E => 1.522 – 0.4 =* **1.122**
- *IG for Feature 2 = Class E – Feature 2 E => 1.522 – 0.551 =* **0.971**

- From the above, IA for Feature 1 is greater hence we use it for the root node. Possible questions include:
  - *If feature 1 = a*
  - *If feature 1 = c*
- Both lead us to a direct label without having to check feature 2. We can choose either but for this illustration the second is chosen as it classifies more instances hence reducing the data size for the next iteration of ID3



ID3 (Iterative Dichotomiser 3) Example 1

- The dataset now updates since two instance have been used in the previous step and classified with minimum impurity

Feature 1	Feature 2	Label
a	*x*	0
b	*x*	0
b	*x*	2

- *Feature 1: 1/3 – a , 2/3 – b*
- *Feature 2: 3/3 – x*
- *Labels: 2/3 – 0 , 1/3 – 2,*

Class / Label Entropy: $-\left(\frac{2}{3}\log_2\left(\frac{2}{3}\right)+\frac{1}{3}\log_2\left(\frac{1}{3}\right)\right)$ ***= 0.918***

Feature 1 Entropy: $\frac{1}{3}\left(-\frac{1}{1}\log_2\left(\frac{1}{1}\right)\right)+\frac{2}{3}\left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right)-\frac{1}{2}\log_2\left(\frac{1}{2}\right)\right)$ ***= 0.667***

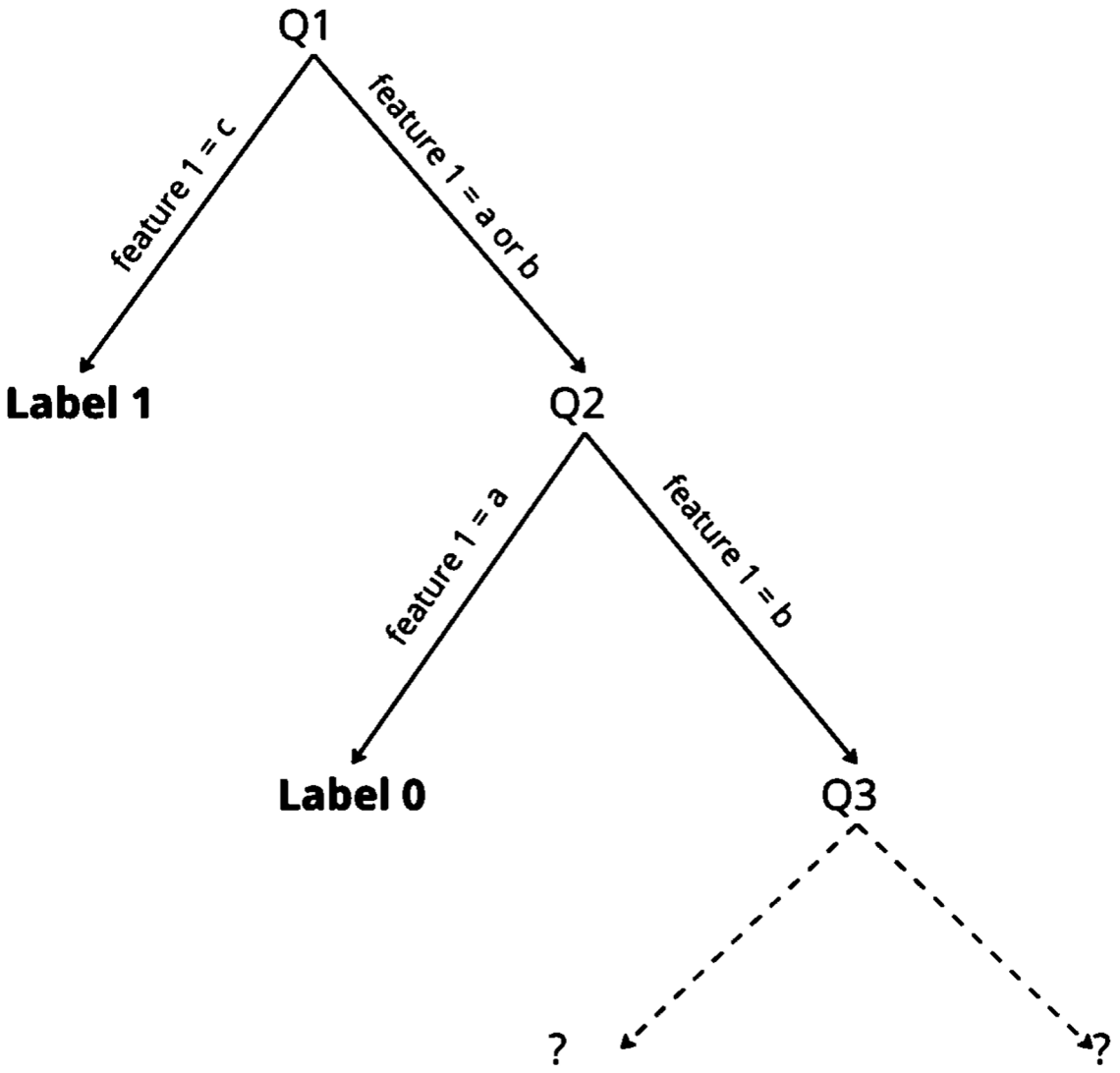
Feature 2 Entropy: $\frac{3}{3}\left(-\frac{2}{3}\log_2\left(\frac{2}{3}\right)-\frac{1}{3}\log_2\left(\frac{1}{3}\right)\right)$ ***= 0.918***

ID3 (Iterative Dichotomiser 3) Example 1

From the previous step, we can now obtain IA for each feature:

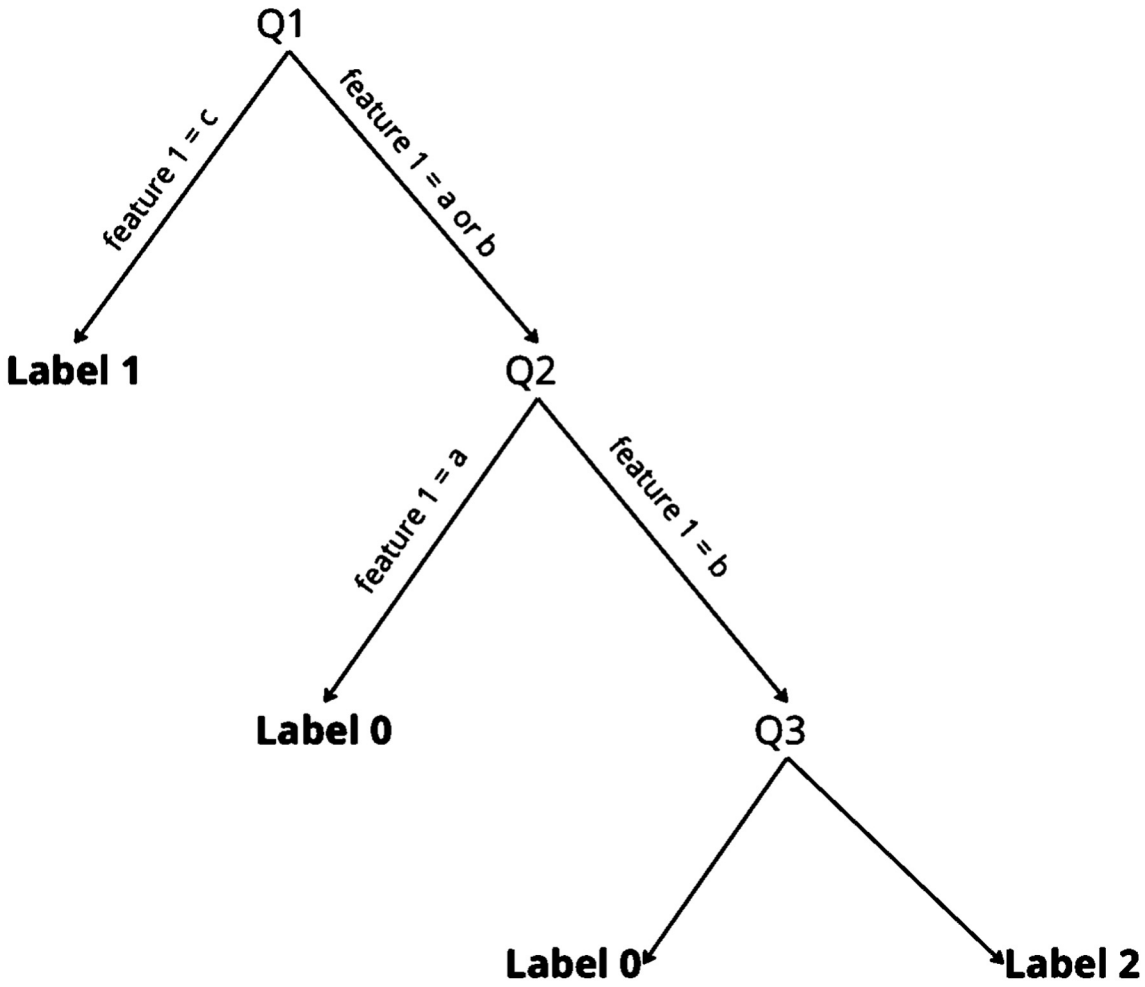
- *IA for Feature 1 = 0.918 – 0.667 = 0.252*
- *IA for Feature 2 = 0.918 – 0.918 = 0*

Feature 1 is clearly greater hence the choice for the next split:



ID3 (Iterative Dichotomiser 3) Example 1

- Since we are only left with two instances with an equal class/label distribution, we can really use Information Gain to decide as their features are also similar.
- As a result we can conclude the tree and specify that instance to be label 0 or 2 has a 50% chance at this node of our decision tree.



ID3 (Iterative Dichotomiser 3) Example 2

- Given the dataset below, design a Decision Tree using ID3

Temperature	Wind	Outlook	Humidity	Outcome
Hot	Weak	Sunny	High	No
Hot	Strong	Sunny	High	No
Hot	Weak	Overcast	High	Yes
Mild	Weak	Rain	High	Yes
Cool	Weak	Rain	Normal	Yes
Cool	Strong	Rain	Normal	No
Cool	Strong	Overcast	Normal	Yes
Mild	Weak	Sunny	High	No
Cool	Weak	Sunny	Normal	Yes
Mild	Weak	Rain	Normal	Yes
Mild	Strong	Sunny	Normal	Yes
Mild	Strong	Overcast	High	Yes
Hot	Weak	Overcast	Normal	Yes
Mild	Strong	Rain	High	No

ID3 (Iterative Dichotomiser 3) Example 2

- Entropy Calculations:

$$\text{Temp} = \frac{4}{14}\left(\frac{-3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right)\right) + \frac{4}{14}\cdot 1 + \frac{6}{14}\left(\frac{-4}{6}\log_2\left(\frac{4}{6}\right) - \frac{2}{6}\log_2\left(\frac{2}{6}\right)\right)$$

$$= 0.9111$$

$$\text{Outlook} = \frac{5}{14}\left(\frac{-3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right)\right) + \frac{4}{14}\cdot 0 + \frac{5}{14}\left(\frac{-2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right)$$

$$= 0.6935$$

$$\text{Wind} = \frac{6}{14}\left(\frac{-3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right)\right) + \frac{8}{14}\left(\frac{-6}{8}\log_2\left(\frac{6}{8}\right) - \frac{2}{8}\log_2\left(\frac{2}{8}\right)\right)$$

$$= 0.8922$$

$$\text{Humidity} = \frac{7}{14}\left(\frac{-3}{7}\log_2\left(\frac{3}{7}\right) - \frac{4}{7}\log_2\left(\frac{4}{7}\right)\right) + \frac{7}{14}\left(\frac{-6}{7}\log_2\left(\frac{6}{7}\right) - \frac{1}{7}\log_2\left(\frac{1}{7}\right)\right)$$

$$= 0.7885$$

$$\text{OGEntropy} = -\left(\frac{9}{14}\log_2\left(\frac{9}{14}\right) + \frac{5}{14}\log_2\left(\frac{5}{14}\right)\right)$$

$$= 0.9403$$

ID3 (Iterative Dichotomiser 3) Example 2

Information Gain Calculations

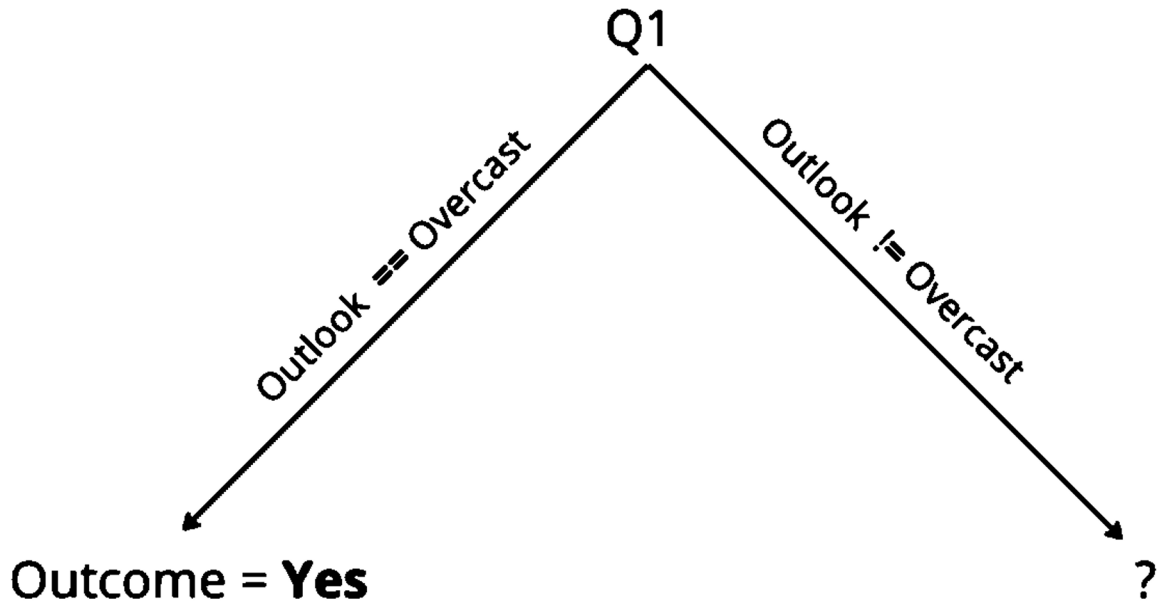
$$\text{TempIG} = \text{OGEntropy} - \text{Temp}$$
$$= 0.0292$$

$$\text{OutlookIG} = \text{OGEntropy} - \text{Outlook}$$
$$= 0.2467$$

$$\text{HumidityIG} = \text{OGEntropy} - \text{Humidity}$$
$$= 0.1518$$

$$\text{WindIG} = \text{OGEntropy} - \text{Wind}$$
$$= 0.0481$$

Sample tree at this point



Continue from this point

Naïve Bayes

Naïve Bayes

- The Naive Bayes Algorithm is one of the crucial algorithms in machine learning that helps with **classification** problems.
- It is derived from Bayes' probability theory and is used for classification, where you train high-dimensional The Naive Bayes Algorithm is known for its simplicity and effectiveness.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes,

  - *Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of colour, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.*
  - *Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.*

- It is faster to build models and make predictions with this algorithm. While creating any ML model, it is better to apply the Bayes theorem
- The theorem states that the probability of an event A happening, given that event B has occurred, can be calculated using the following formula:

Naïve Bayes

- The algorithm is based on the Bayes theorem:

$$P(x|y) = \frac{P(y|x) * P(x)}{P(y)}$$

  - P(x|y) denotes how event x happens when event y takes place. (posterior probability)
  - P(y|x) represents how often event y happens when event x takes place first. (likelihood)
  - P(x) represents the probability of event x happening on its own. (prior probability)
  - P(y) represents the probability of event y happening on its own. (marginal probability)

- *The Bayes Rule is a method for determining P(x|y) from P(y|x). In short, it provides you with a way of calculating the probability of a hypothesis with the provided evidence.*

Naïve Bayes

- Some best examples of the Naive Bayes Algorithm are:
    - *Sentimental analysis,*
    - *Classifying new articles,*
    - *Spam filtration and,*
    - *Fault Diagnosis*

Advantages

i. Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.

ii. It can be used for Binary as well as Multi-class Classifications.

iii. It performs well in Multi-class predictions as compared to the other Algorithms.

iv. It is the most popular choice for text classification problems.

v. It works well with high-dimensional data

Disadvantages

i. The assumption that all features are independent is not usually the case in real life so it makes naive bayes algorithm less accurate than complicated algorithms. Speed comes at a cost!

Naïve Bayes: Types

There are three types of Naive Bayes Model, which are given below:

1. Gaussian: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.

2. Multinomial: Is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.

3. Bernoulli: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Naïve Bayes: Example

- Suppose we have a dataset of texts and corresponding labels to perform sentiment analysis for a new unlabelled text *"I am great."* e.g.

Sentence	Label
I am happy	*positive*
This is sad	*negative*
It's great	*postive*

Step 1: Build the vocabulary: ["i", "am", "happy", "this", "is", "sad", "it's", "great"]

Step 2: Calculate class priors

We have two categories: "positive" and "negative". In our dataset, we have 2 positive sentences and 1 negative sentence. So the class priors are:

- $P(positive) = 2/3 \approx 0.67$
- $P(negative) = 1/3 \approx 0.33$

Naïve Bayes: Example

Step 3: Calculate word likelihoods.

- For each word in the vocabulary, calculate the likelihood of observing that word in each class. We count the occurrences of each word in each class and divide by the total number of words in that class.

Word	P(Word|positive)	P(Word|negative)
i	1/5 = 0.2	0/3 = 0
am	1/5 = 0.2	0/3 =0
happy	1/5 = 0.2	0/3 =0
this	0/5 = 0	1/3 ≈ 0.33
is	0/5 = 0	1/3 ≈ 0.33
sad	0/5 = 0	1/3 ≈ 0.33
it’s	1/5 = 0.2	0/3 = 0
great	1/5 = 0.2	0/3 = 0

Naïve Bayes: Example

Step 4: Calculate posterior probabilities

- For each class, calculate the posterior probability of the test sentence belonging to that class. Multiply the prior probability of the class with the conditional probability of observing each word in the sentence, given the class.

- **For the positive class:**

  $P(positive|sentence) \propto P(positive) * P(i|positive) * P(am|positive) * P(great|positive)$
  $\approx 0.67 * 0.2 * 0.2 * 0.2$
  ≈ 0.0085

- **For the negative class:**

  $P(negative|sentence) \propto P(negative) * P(i|negative) * P(am|negative) * P(great|negative)$

  $\approx 0.33 * 0 * 0 * 0$

  $= 0$

Step 5: Make the classification decision

- The posterior probability for the positive class is approximately 0.0085, while the posterior probability for the negative class is 0. Since the posterior probability for the positive class is higher, we can classify the test sentence "I am great" as "positive".

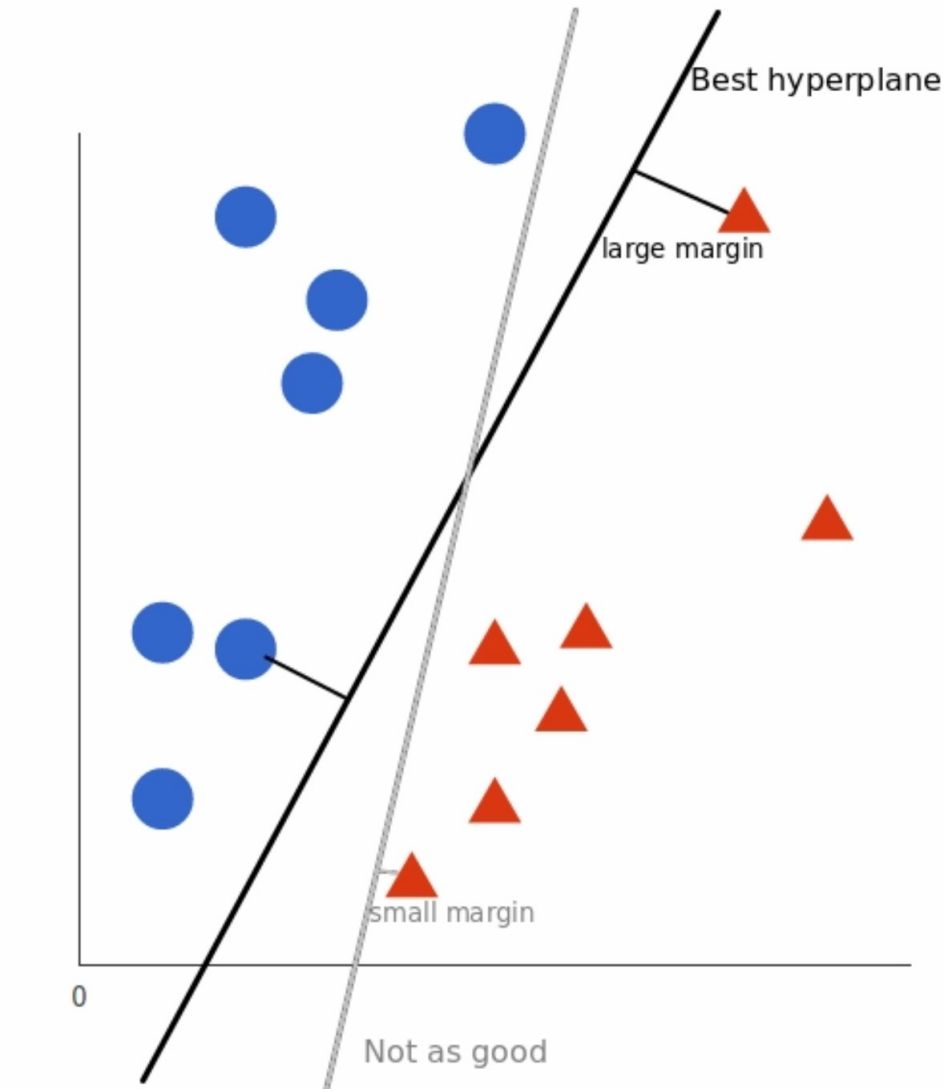
- Therefore, according to the Naive Bayes classifier, the test sentence "I am great" is classified as "positive".

Support Vector Machines (SVMs)

SVMs

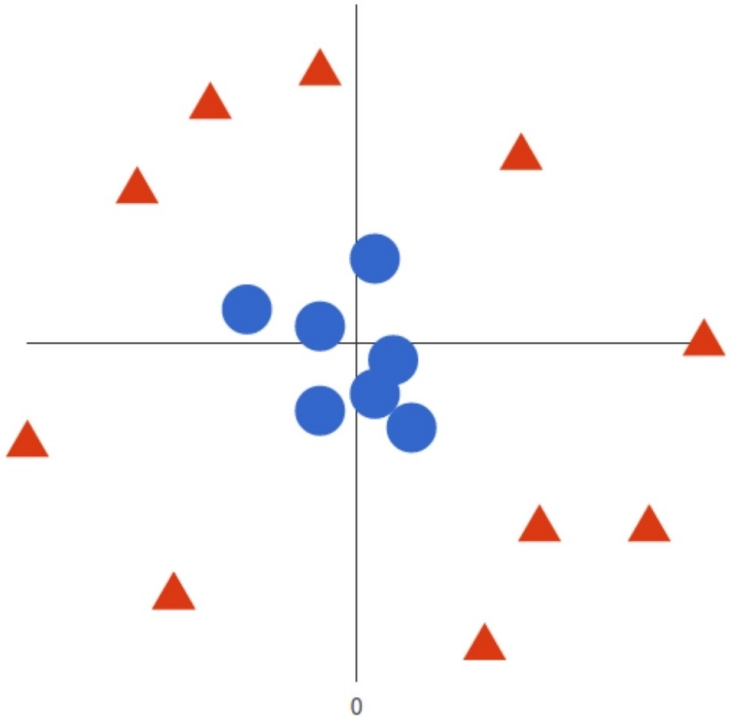
- A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text.
- Compared to newer algorithms like neural networks, they have two main advantages: higher speed and better performance with a limited number of samples (in the thousands). This makes the algorithm very suitable for text classification problems, where it's common to have access to a dataset of at most a couple of thousands of tagged samples.
- A simple linear SVM classifier works by making a straight line between two classes. That means all of the data points on one side of the line will represent a category and the data points on the other side of the line will be put into a different category. This means there can be an infinite number of lines to choose from.
- What makes the linear SVM algorithm better than some of the other algorithms, like k-nearest neighbors, is that it chooses the best line to classify your data points. It chooses the line that separates the data and is the furthest away from the closet data points as possible.

SVMs: 2-D example

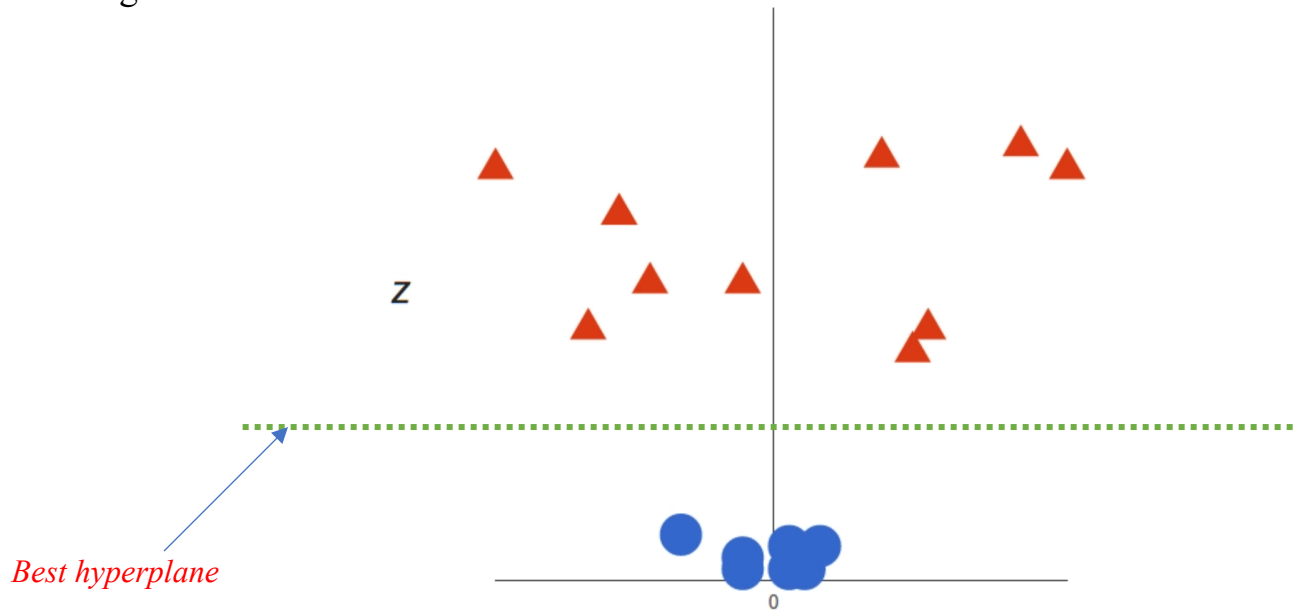


- Anything that falls to one side of it we will classify as blue, and anything that falls to the other as red.
- We're trying to separate these data points by the category they should fit in, but we don't want to have any data in the wrong category.
- So the two closest data points give the support vectors which we use to find the optimal boundary (Best hyperplane)

SVMs: 3D example (Non-linear)



- In this illustration (**Left**), our data isn't linear as the previous example. SVM can still work and identify an optimal boundary however it is necessarily going to be in this dimension.
- We create a new z dimension, and we rule that it be calculated a certain way that is convenient for us: $z = x^2 + y^2$ *(equation for a circle)*.
- This will give us a three-dimensional as shown **below**



SVMs: Types, Pros & Cons

Types:

- Simple SVM: Typically used for linear classification problems.
- Kernel SVM: Has more flexibility for non-linear data because you can add more features to fit a hyperplane instead of a two-dimensional space.

Pros:

i. Effective on datasets with multiple features, like financial or medical data.

ii. Effective in cases where number of features is greater than the number of data points.

iii. Uses a subset of training points in the decision function called support vectors which makes it memory efficient.

iv. Different kernel functions can be specified for the decision function.

Cons:

i. If the number of features is a lot bigger than the number of data points, avoiding over-fitting when choosing kernel functions and regularization term is crucial.

ii. SVMs don't directly provide probability estimates. Those are calculated using an expensive five-fold cross-validation.

iii. Works best on small sample sets because of its high training time.

Kernel Functions

Linear

- These are commonly recommended for text classification because most of these types of classification problems are linearly separable.
- Imagine you have a toy train track that is completely straight. The linear kernel function behaves like a ruler, measuring the distances between points on the track. It works well when the data can be separated by a straight line.
- The linear kernel works really well when there are a lot of features, and text classification problems have a lot of features. Linear kernel functions are faster than most of the others and you have fewer parameters to optimize.
- Here's the function that defines the linear kernel:

$$f(X) = w^T * X + b$$

- In this equation, w is the weight vector that you want to minimize, X is the data that you're trying to classify, and b is the linear coefficient estimated from the training data. This equation defines the decision boundary that the SVM returns.

Kernel Functions

Polynomial

- The polynomial kernel isn't used in practice very often because it isn't as computationally efficient as other kernels and its predictions aren't as accurate.
- Imagine you have a set of colored building blocks, and you want to classify them based on their shapes. The polynomial kernel function uses the concept of polynomial expressions, which can represent curves of different shapes. It allows you to capture more complex relationships between the data points.
- Here's the function for a polynomial kernel:

$$f(X1, X2) = (a + X1^T * X2) ^ b$$

- This is one of the more simple polynomial kernel equations you can use. f(X1, X2) represents the polynomial decision boundary that will separate your data. X1 and X2 represent your data.

Kernel Functions

Gaussian Radial Basis Function (RBF)

- One of the most powerful and commonly used kernels in SVMs. Usually the choice for non-linear data.
- Imagine you have a jar filled with differently sized rubber balls. The Gaussian kernel function measures the similarity between two balls by comparing their distance in a three-dimensional space. The closer the balls are, the more similar they are considered to be. It is useful when the decision boundary between classes is non-linear and can take various shapes.
- Here's the equation for an RBF kernel:

$$f(X1, X2) = exp(-gamma * ||X1 - X2||^2)$$

- In this equation, gamma specifies how much a single training point has on the other data points around it. ||X1 - X2|| is the dot product between your features.

Kernel Functions

Sigmoid

- More useful in neural networks than in support vector machines, but there are occasional specific use cases.
- Imagine you have a hotdog and a hamburger, and you want to classify them based on their taste. The sigmoid kernel function mimics the behavior of a taste tester who evaluates the flavors based on their similarities or differences. It can capture non-linear relationships and is commonly used in neural networks.
- Here's the function for a sigmoid kernel:

$$f(X, y) = tanh(alpha * X^T * y + C)$$

- In this function, alpha is a weight vector and C is an offset value to account for some mis-classification of data that can happen.

SVMs: Uses / Justifications

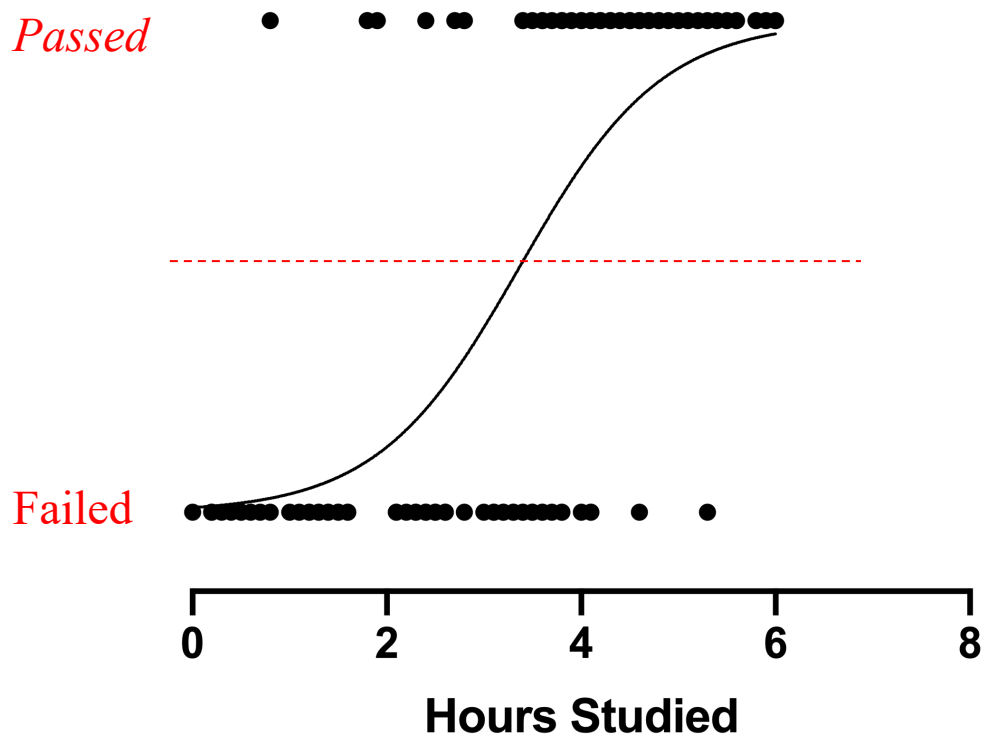
- SVMs are used in applications like handwriting recognition, intrusion detection, face detection, email classification, gene classification, and in web pages. This is one of the reasons we use SVMs in machine learning. It can handle both classification and regression on linear and non-linear data.

- Another reason we use SVMs is because they can find complex relationships between your data without you needing to do a lot of transformations on your own. It's a great option when you are working with smaller datasets that have tens to hundreds of thousands of features. They typically find more accurate results when compared to other algorithms because of their ability to handle small, complex datasets.

Logistic Regression

Logistic Regression

- Logistic Regression is a kind of parametric classification model, despite having the word ‘regression’ in its name.
- This means that logistic regression models are models that have a certain fixed number of parameters that depend on the number of input features, and they output categorical prediction, like for example if a plant belongs to a certain species or not.
- In reality, the theory behind Logistic Regression is very similar to the one from Linear Regression where we fit a straight line to datapoints.
- In Logistic Regression, we don’t directly fit a straight line to our data like in linear regression. Instead, we fit a S shaped curve, called Sigmoid, to our observations.



Log. Reg

- The logistic regression function is defined as:

$$\sigma(z) = 1 / (1 + e^{(-z)})$$

- In this equation, σ(z) represents the output or the predicted probability, and z represents the input to the function, which is a linear combination of the input variables and their corresponding weights.

- The logistic function maps the linear combination to a value between 0 and 1, which can be interpreted as the probability of the event occurring.

- The logistic function has an S-shaped curve, with the following properties:

    - As z approaches positive infinity, σ(z) approaches 1, indicating a high probability of the event occurring.

    - As z approaches negative infinity, σ(z) approaches 0, indicating a low probability of the event occurring.

    - At z = 0, σ(z) equals 0.5, implying an equal probability of the event occurring or not occurring.

Log. Reg

- In logistic regression, the input variables are multiplied by their respective weights, and the resulting products are summed up to calculate the value of z. The logistic function then transforms this value to obtain the predicted probability.
- During the training process, the logistic regression model adjusts the weights through a process called optimization, typically using an algorithm like maximum likelihood estimation or gradient descent.
- The objective is to find the set of weights that maximizes the likelihood of the observed data given the model.
- Once the model is trained and the weights are determined, the logistic regression function can be used to make predictions by computing the probability of the event occurring for new input data.
- A threshold can be applied to convert the predicted probabilities into binary predictions, depending on the specific application or decision-making criteria.

Log. Reg

- Let's consider a simple example where we have a logistic regression model trained to predict whether a student will pass (1) or fail (0) an exam based on the number of hours they studied. We'll assume the model has the following weights:
- Weight (w) = 0.5 Bias (b) = -2.5
- Now, let's say we want to predict the probability of a student passing the exam after studying for 7 hours. We will use the logistic regression function to calculate the probability step by step:
- Step 1: Calculate the linear combination (z)

  $z = w * x + b$

  where: w = weight x = input (number of hours studied) b = bias

  In our case: w = 0.5 x = 7 b = -2.5

  $z = 0.5 * 7 + (-2.5) = 3.5 - 2.5 = 1$

Log. Reg

- Step 2: Apply the logistic regression function

  $\sigma(z) = 1 / (1 + e^{(-z)})$

  In our case: $z = 1$

  $\sigma(z) = 1 / (1 + e^{(-1)}) = 1 / (1 + 0.3679) = 1 / 1.3679 \approx 0.731$

- So, the calculated probability using logistic regression is approximately 0.731. This implies that the model predicts a 73.1% chance that the student will pass the exam after studying for 7 hours.

- It's important to note that in practice, the weights and bias are determined through the training process, which involves optimizing the model's parameters to best fit the training data. This was a simplified example for illustration purposes.

- In the logistic regression function, the term "e" refers to Euler's number, also known as the base of the natural logarithm. Euler's number is a mathematical constant approximately equal to 2.71828.

Log. Reg

- To derive the weights (w) and the bias (b) from a logistic regression dataset, you typically need to perform a process called model training or parameter estimation. There are different methods for estimating the parameters, but one commonly used approach is maximum likelihood estimation (MLE).

1. Data Preparation: Ensure your dataset is properly prepared for logistic regression. This involves having a target variable (the binary outcome you want to predict, such as pass/fail) and one or more input features (the variables you will use to predict the outcome). It's also important to preprocess and normalize the data if needed.

2. Model Setup: Define the logistic regression model, including the choice of features, the logistic regression function, and the hypothesis to be tested.

3. Define the Likelihood Function: In logistic regression, the likelihood function measures the probability of obtaining the observed data given a specific set of model parameters (weights and bias). The goal is to find the parameters that maximize this likelihood.

4. Optimization: To find the optimal values for the weights and bias, you can use optimization algorithms such as gradient descent or specialized solvers. These methods iteratively update the parameters based on the gradients of the likelihood function until convergence is achieved.

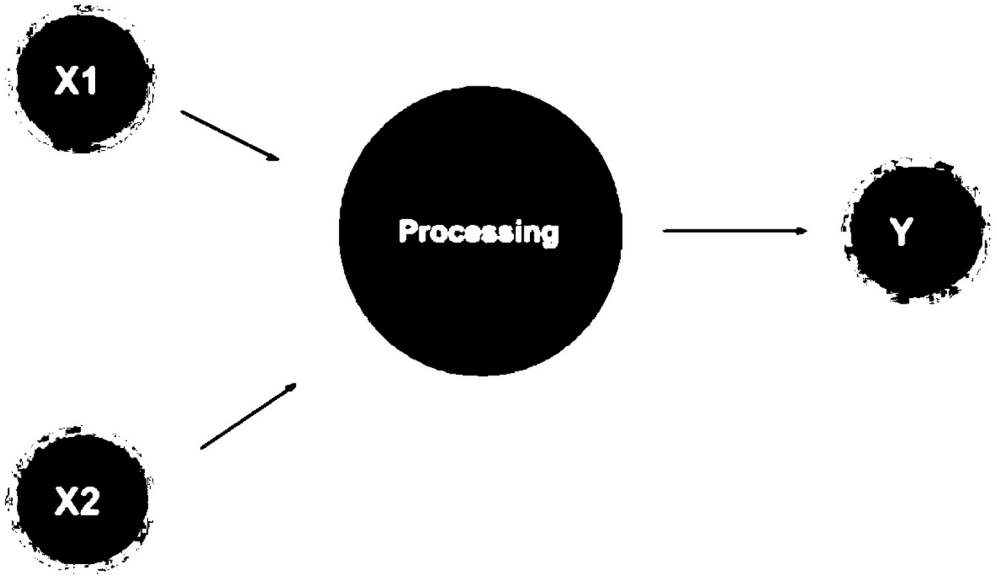
5. Evaluate the Model: Once the parameters are estimated, you should evaluate the performance of the model using evaluation metrics such as accuracy, precision, recall, or ROC curves. This step helps assess the model's ability to generalize and make predictions on unseen data.

Perceptron

Neural Network Perceptron

- A single-layer perceptron is the basic unit of a neural network. A perceptron consists of input values, weights and a bias, a weighted sum and activation function. The perceptron was first introduced by American psychologist, Frank Rosenblatt in 1957 at Cornell Aeronautical Laboratory (here is a link to the original paper if you are interested).

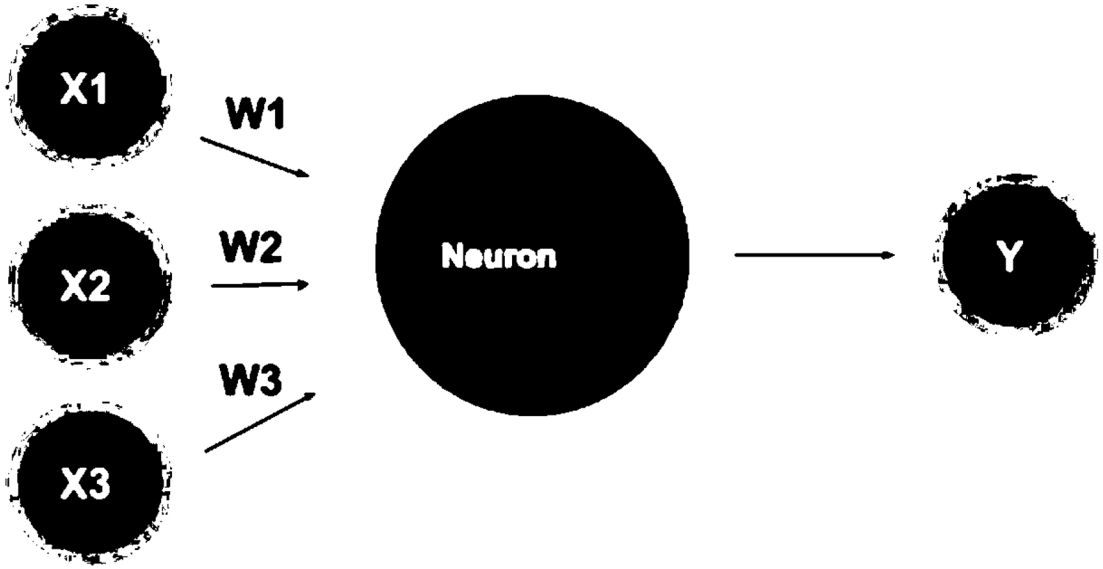
- Rosenblatt was heavily inspired by the biological neuron and its ability to learn. Rosenblatt's perceptron consists of one or more inputs, a processor, and only one output.



Neural Network Perceptron

- A perceptron works by taking in some numerical inputs along with what is known as weights and a bias. It then multiplies these inputs with the respective weights(this is known as the weighted sum). These products are then added together along with the bias. The activation function takes the weighted sum and the bias as inputs and returns a final output.

- Assume we have a single neuron and three inputs x1, x2, x3 multiplied by the weights w1, w2, w3 respectively



Activation Functions

$$y - x1w1 + x2w2 + x3w3....$$

- This function is called the weighted sum because it is the sum of the weights and inputs. This looks like a good function, but if we wanted the outputs to fall into a certain range say 0 to 1; we introduce an activation function.
- An activation function is a function that converts the input given (the input, in this case, would be the weighted sum) into a certain output based on a set of rules. There are different kinds of activation functions that exist, for example:
  - *Hyperbolic Tangent: used to output a number from -1 to 1.*
  - *Sigmoid Function: used to output a number from 0 to 1.*
- They introduce non-linearity to the network, allowing it to learn and represent complex relationships in the data. Activation functions are applied to the weighted sum of inputs in each neuron to produce the neuron's output or activation value.

Bias & Output

Bias

- A bias term is often added to the perceptron, which acts as an adjustable constant that shifts the activation function's threshold. It allows the perceptron to make decisions even when all input signals are zero. Think of it as some *threshold the perceptron must reach before the output is produced*.

- We thus end up with a function as:

$$y = \sum(Weights * Inputs) + Bias$$

- Notice that the activation function takes in the **weighted sum plus the bias** as inputs to create a single output.

Output

- The output of the perceptron is the result of the activation function applied to the weighted sum plus the bias. It can be a binary output (0 or 1) or a continuous output between 0 and 1, depending on the chosen activation function.

Perceptron Learning Rule

- The perceptron learning rule, also known as the delta rule or delta learning rule, is an algorithm used to train perceptrons in supervised learning settings. It updates the weights of the perceptron based on the prediction error, aiming to minimize the difference between the predicted output and the desired output. Here's an overview steps of the perceptron learning rule:

    1. *Initialization:* Initialize the weights (w) and bias (b) to small random values or zeros.
    2. *Training Data*: Obtain a set of training data, consisting of input vectors (x) and their corresponding target outputs (y).
    3. *Forward Propagation*: For each input vector (x), compute the weighted sum (net input) and apply the activation function to obtain the predicted output (y_pred). net input = $(w \cdot x) + b$ y_pred = activation_function(net input)
    4. *Error Calculation*: Calculate the prediction error (delta) by subtracting the predicted output from the target output. delta = y - y_pred
    5. *Weight Update*: Update the weights and bias using the perceptron learning rule: Δw = learning_rate * delta * x Δb = learning_rate * delta where Δw is the change in weights, learning_rate is a positive constant determining the step size of the update, and x is the input vector. *Update the weights and bias* by adding the respective deltas: w_new = w_old + Δw b_new = b_old + Δb
    6. *Repeat*: Repeat steps 3 to 5 for all training samples or until convergence criteria are met.
    7. *Convergence* Criteria: Convergence can be determined by various conditions, such as reaching a maximum number of iterations, achieving a desired level of accuracy, or when the weights and bias no longer change significantly.
    8. *Evaluation:* After training, evaluate the performance of the perceptron on unseen data to assess its generalization capability.

Convergence

- Imagine you have a game where you need to guess the correct answer. At first, you might make random guesses, but as you receive feedback, you start to adjust your guesses to get closer to the correct answer. Eventually, you get better and better at guessing until you finally guess the correct answer every time.
- In machine learning, convergence is similar to this game. When we train a model, like a perceptron, it starts by making random guesses or predictions. Then, it receives feedback on how accurate its predictions are compared to the correct answers. Based on this feedback, the model adjusts itself to make better predictions.
- Convergence *means that the model's predictions get closer and closer to the correct answers as it continues to learn from the feedback*. It's like getting better at the guessing game by making more accurate guesses. The model keeps making adjustments until it reaches a point where its predictions are very close to the correct answers, and it doesn't need to make significant changes anymore.
- So, when we say that a model has converged, it means that it has reached a state where its predictions are very accurate and stable. It has learned from the data and can make reliable predictions or classifications based on what it has learned.
- Convergence is an essential goal in machine learning because it indicates that the model has successfully learned from the data and can be used to make accurate predictions on new, unseen data.

Ensemble Modelling

Ensemble Modelling

Ensemble modelling is a technique in machine learning where multiple models are combined to improve predictive performance and generalization. There are several popular algorithms used for building ensemble models:

- Bagging (Bootstrap Aggregating): Bagging involves training multiple models independently on different subsets of the training data, typically obtained by resampling with replacement. The final prediction is obtained by aggregating the predictions of all models. Examples of bagging algorithms include Random Forest and Extra Trees.
- Boosting: Boosting algorithms train multiple models sequentially, where each subsequent model focuses on correcting the mistakes made by the previous models. The final prediction is obtained by combining the predictions of all models, often weighted based on their performance. Examples of boosting algorithms include AdaBoost, Gradient Boosting, and XGBoost.
- Stacking: Stacking combines multiple models by training a meta-model that takes the predictions of the individual models as inputs. The meta-model learns to make predictions based on the outputs of the base models. Stacking can be done in multiple levels, where the predictions of one level are used as inputs for the next level. Stacking allows models to complement each other's strengths and compensate for weaknesses.

Ensemble Modelling

- Voting: Voting ensembles combine the predictions of multiple models by majority voting or weighted voting. In majority voting, the class predicted by the majority of models is chosen as the final prediction. In weighted voting, each model's prediction is assigned a weight, and the final prediction is computed based on the weighted sum of predictions.
- Random Subspace Method: The random subspace method, also known as feature bagging, involves training models on different subsets of the input features. Each model focuses on a different set of features, and their predictions are combined to make the final prediction. This technique is particularly useful when dealing with high-dimensional data.
- These are just a few examples of ensemble algorithms commonly used in machine learning. Each algorithm has its own advantages and is suited for different types of problems. The choice of ensemble algorithm depends on the characteristics of the data, the base models being used, and the desired performance objectives.

Unsupervised Learning

Unsupervised Learning

- Unsupervised learning is a type of machine learning where an algorithm learns patterns or structures in the data without explicit guidance or labelled examples. Unlike supervised learning, where the algorithm is provided with labeled data to learn from, unsupervised learning seeks to discover intrinsic patterns, relationships, or groupings within the data on its own.
- The primary goal of unsupervised learning is to extract useful information, insights, or representations from the input data without any prior knowledge or predefined categories. It is often used for tasks such as clustering, dimensionality reduction, anomaly detection, and generative modelling.
- Unsupervised learning is often used for tasks such as

    1. *clustering,*
    2. *dimensionality reduction, and*
    3. *anomaly detection*

Clustering

- Clustering is the task of grouping data points together based on their similarity. There are many different clustering algorithms, each with its own strengths and weaknesses. Some popular clustering algorithms include:
- K-means clustering: This algorithm divides the data into k clusters, where k is a user-defined parameter. The algorithm starts by randomly assigning each data point to a cluster. It then iteratively updates the cluster centroids and assigns each data point to the cluster with the closest centroid. This process continues until the cluster centroids no longer change significantly.
- Hierarchical clustering: This algorithm builds a hierarchy of clusters by repeatedly merging similar clusters. The algorithm can be either agglomerative (bottom-up) or divisive (top-down).
- Spectral clustering: This algorithm uses the spectrum of the data's similarity matrix to cluster the data.

Dimensionality reduction

- Dimensionality reduction is the task of reducing the number of features in a dataset while preserving as much information as possible. This can be useful for tasks such as visualization and machine learning.
- There are many different dimensionality reduction algorithms, each with its own strengths and weaknesses. Some popular dimensionality reduction algorithms include:
- Principal component analysis (PCA): This algorithm projects the data onto a lower-dimensional subspace that captures the most variance in the data.
- Singular value decomposition (SVD): This algorithm decomposes the data matrix into three matrices: a left singular matrix, a singular value matrix, and a right singular matrix. The singular value matrix can be used to project the data onto a lower-dimensional subspace.
- Non-negative matrix factorization (NMF): This algorithm decomposes the data matrix into two matrices: a non-negative basis matrix and a non-negative coefficient matrix. The basis matrix can be used to represent the data in a lower-dimensional subspace.

Anomaly detection

- Anomaly detection is the task of identifying data points that are significantly different from the rest of the data. This can be useful for tasks such as fraud detection and quality control.
- There are many different anomaly detection algorithms, each with its own strengths and weaknesses. Some popular anomaly detection algorithms include:
- Isolation forest: This algorithm builds a forest of decision trees and then identifies data points that are isolated from the rest of the trees.
- Local outlier factor (LOF): This algorithm calculates the local density of each data point and then identifies data points that have a low local density.
- One-class support vector machine (OCSVM): This algorithm trains a support vector machine to separate the normal data points from the rest of the space. Data points that are on the wrong side of the decision boundary are considered to be anomalies.

Applications

- Unsupervised learning is used in a wide variety of applications, including:

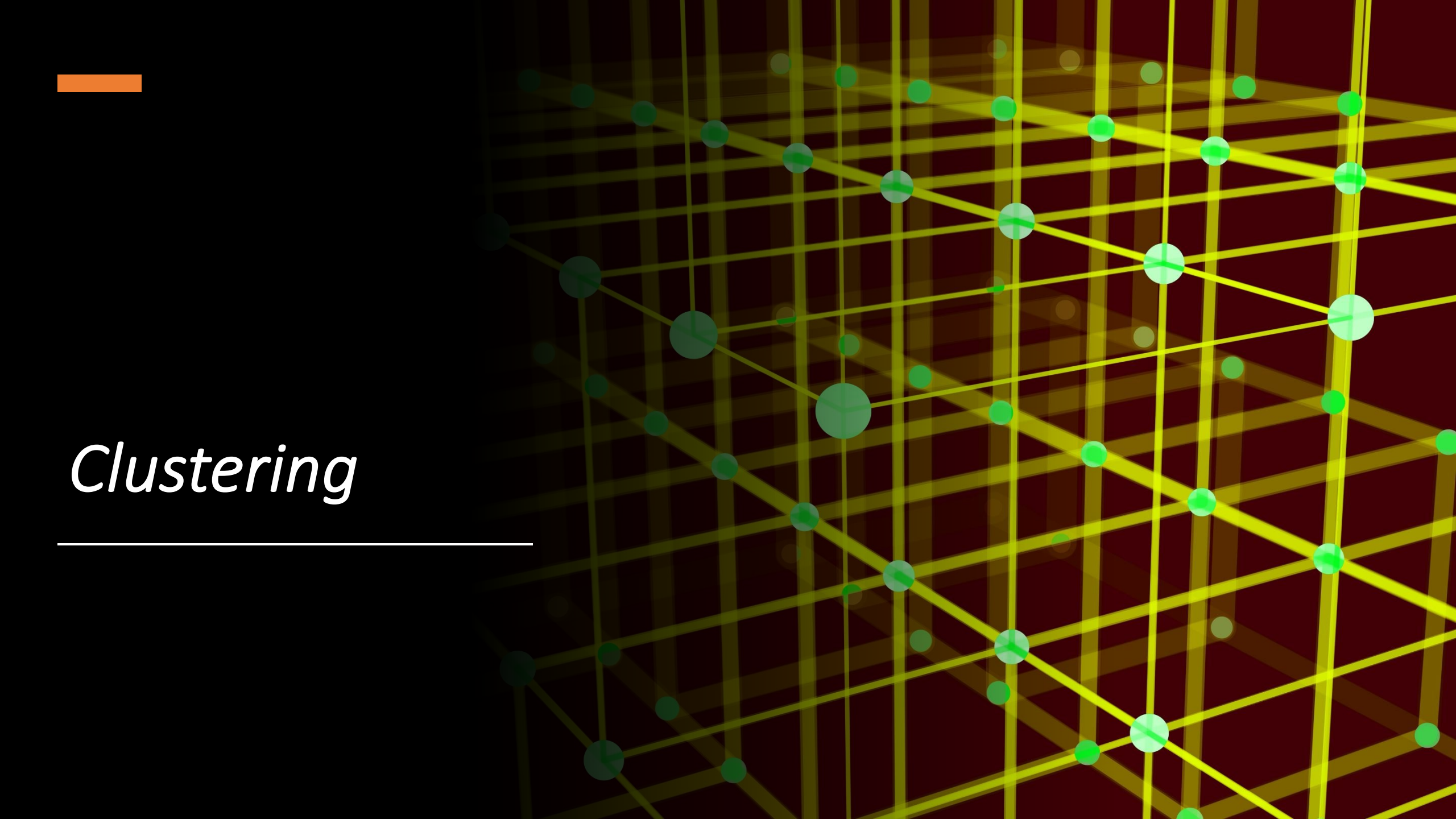
1. *Customer segmentation*: Unsupervised learning can be used to segment customers into groups based on their purchase behavior. This information can then be used to target marketing campaigns more effectively.

2. *Product recommendation*: Unsupervised learning can be used to recommend products to customers based on their purchase history and other factors.

3. *Fraud detection*: Unsupervised learning can be used to identify fraudulent transactions by looking for patterns that are inconsistent with normal behavior.

4. *Quality control*: Unsupervised learning can be used to identify defective products by looking for patterns that are inconsistent with the expected quality.

Clustering



K-means Clustering

- K-means clustering is a popular algorithm for grouping data points into clusters. The algorithm works by iteratively assigning data points to clusters based on their distance to the cluster centroids. The centroid of a cluster is the average of all the data points in the cluster. The algorithm starts by randomly assigning data points to clusters. It then iteratively updates the cluster centroids and assigns each data point to the cluster with the closest centroid. This process continues until the cluster centroids no longer change significantly.
- K-means clustering is a simple and efficient algorithm that can be used to cluster data points in a variety of dimensions. However, it is important to note that K-means clustering is not a perfect algorithm. It can be sensitive to the choice of the number of clusters, and it can be fooled by outliers.

Advantages

- It is a simple and efficient algorithm.
- It can be used to cluster data points in a variety of dimensions.
- It is relatively robust to noise.

Disadvantages

- It can be sensitive to the choice of the number of clusters.
- It can be fooled by outliers.

K-means
Clustering
Example

Voronoi Clustering

- Voronoi clustering is a type of clustering algorithm that groups data points based on their nearest neighbors. The algorithm works by creating a Voronoi diagram, which is a partition of the space into cells such that each cell contains only points that are closer to that cell's center than any other cell's center. The data points are then grouped together based on the cell that they belong to.
- Voronoi clustering is a relatively simple algorithm, but it can be very effective for a variety of clustering tasks. It is particularly well-suited for tasks where the data points are not well-separated, as it can identify clusters that are not easily detected by other clustering algorithms.

Advantages

- It is a simple and efficient algorithm.
- It can identify clusters that are not easily detected by other clustering algorithms.
- It is robust to noise.

Disadvantages

- It can be sensitive to the choice of parameters.
- It can be difficult to interpret the results.

Voronoi Clustering

- Voronoi clustering is a powerful tool that can be used for a variety of clustering tasks. It is simple to implement and can be used to identify clusters that are not easily detected by other clustering algorithms. However, it is important to be aware of the algorithm's limitations before using it.

Uses

1. Customer segmentation: Voronoi clustering can be used to segment customers into groups based on their purchase behavior. This information can then be used to target marketing campaigns more effectively.
2. Product recommendation: Voronoi clustering can be used to recommend products to customers based on their purchase history and other factors.
3. Fraud detection: Voronoi clustering can be used to identify fraudulent transactions by looking for patterns that are inconsistent with normal behavior.
4. Quality control: Voronoi clustering can be used to identify defective products by looking for patterns that are inconsistent with the expected quality.


Dimension Reduction

Association Rule Mining

Association Rule Mining

- Association rule mining is a data mining technique that finds frequent itemsets in a large dataset. It is a rule-based method that discovers relationships between items in a dataset. Association rule mining is often used in market basket analysis to find items that are often bought together.
- An association rule has two parts: an antecedent and a consequent. The antecedent is the set of items that must be present in a transaction for the rule to be true. The consequent is the set of items that are likely to be present in a transaction if the antecedent is present.
- The support of an association rule is the percentage of transactions in the dataset that contain the antecedent and the consequent. The confidence of an association rule is the probability that a transaction that contains the antecedent will also contain the consequent.
- Association rule mining algorithms typically use a two-step process to find association rules:

i. *Find all frequent item-sets in the dataset.*

ii. *Generate association rules from the frequent item-sets.*

Association Rule Mining

- The first step is to find all frequent item-sets in the dataset. This is done by using a frequent itemset mining algorithm. A frequent itemset mining algorithm scans the dataset and finds all item-sets that occur in at least a minimum support percentage of transactions.
- The second step is to generate association rules from the frequent item-sets. This is done by using an association rule mining algorithm. An association rule mining algorithm scans the frequent item-sets and generates all association rules that have a minimum confidence percentage.
- The support and confidence thresholds are two important parameters that control the results of association rule mining. The support threshold determines how frequently an itemset must occur in the dataset in order to be considered frequent. The confidence threshold determines how likely it is that a consequent item will be present in a transaction if the antecedent item is present.
- Association rule mining can be used to find a variety of interesting patterns in a dataset. For example, it can be used to find items that are often bought together, products that are often purchased by the same customers, and websites that are often visited by the same users.

Association Rule Mining

Association rule mining is a powerful tool that can be used to gain insights into customer behavior and to improve the efficiency of business processes. Some of the benefits of using association rule mining:

- It can help you to identify patterns in your data that you may not have noticed otherwise.
- It can help you to make better decisions about your business, such as what products to sell, what marketing campaigns to run, and how to improve your customer service.
- It can help you to save money by identifying areas where you can improve efficiency.

Association Rule Mining

Benefits:

1. Association rule mining algorithms can be used to discover hidden patterns in large databases.
2. Association rule mining algorithms can be used to identify potential marketing opportunities.
3. Association rule mining algorithms can be used to detect fraudulent activity.
4. Association rule mining algorithms can be used to gain insights into customer behaviour.

Challenges:

1. Association rule mining algorithms can be computationally expensive.
2. Association rule mining algorithms can be sensitive to the choice of the minimum support and minimum confidence parameters.
3. Association rule mining algorithms can only discover association rules, not causation.

Apriori Algorithm

- Apriori is a frequent itemset mining algorithm that was first proposed by Agrawal et al. in 1994. It is a very efficient algorithm for finding all frequent item-sets in a large dataset. The Apriori algorithm works by iteratively finding all frequent item-sets of increasing size.
- The Apriori algorithm works as follows:
    1. Start with the set of all 1-itemsets.
    2. For each k-itemset, check if it is frequent. If it is, add it to the set of frequent itemsets.
    3. Generate all (k+1)-item-sets by joining k- item-sets together.
    4. Repeat steps 2 and 3 until no new frequent item-sets are found.
- The Apriori algorithm uses a number of pruning techniques to reduce the number of candidate item-sets that need to be checked. These pruning techniques are based on the following two principles:
    i. The Apriori principle: An itemset cannot be frequent if any of its subsets are not frequent.
    ii. The downward closure property: If an itemset is frequent, then all of its subsets are also frequent.

Apriori Algorithm

The Apriori algorithm is a very efficient algorithm for finding frequent item-sets. However, it can be computationally expensive for large datasets. There are a number of other frequent itemset mining algorithms that are more efficient for large datasets. These algorithms include Eclat and FPGrowth.

Advantages

i. It is a very efficient algorithm for finding frequent item-sets.

ii. It is a simple algorithm to understand and implement.

iii. It is a very versatile algorithm that can be used in a variety of different applications.

Disadvantages

i. It can be computationally expensive for large datasets.

ii. It can be difficult to find the right parameters for the Apriori algorithm.

iii. The Apriori algorithm can generate a large number of frequent item-sets, which can be difficult to analyze.

Apriori Algorithm

Let's consider a simple example of basket analysis using the Apriori algorithm. Imagine you have a dataset of customer transactions from a grocery store. Each transaction consists of a list of items purchased by a customer.

Here's a sample dataset (transaction):

i. *Transaction 1: {Bread, Milk, Eggs}*

ii. *Transaction 2: {Milk, Butter}*

iii. *Transaction 3: {Bread, Milk, Butter}*

iv. *Transaction 4: {Bread, Eggs}*

v. *Transaction 5: {Milk, Eggs}*

vi. *Transaction 6: {Bread, Milk, Eggs, Butter}*

Apply the Apriori algorithm to find frequent item-sets and generate association rules.

Question

Consider the following dataset representing customer transactions:

Transaction 1: {bread, milk, eggs}
Transaction 2: {bread, diapers}
Transaction 3: {milk, diapers}
Transaction 4: {bread, milk, diapers}
Transaction 5: {bread, diapers}

Apply the Apriori algorithm with a minimum support threshold of 40% and a minimum count of 2 to mine frequent itemsets and association rules.

1. Calculate the support for each item and identify the frequent itemsets of size 1.
2. Generate frequent itemsets of size 2 and calculate their support.
3. Generate frequent itemsets of size 3 and calculate their support.
4. Choose an association rule and calculate its confidence.
5. Explain your findings and the implications of the association rule.

Note: Show your calculations and provide a clear explanation of each step.

This question tests your understanding of the Apriori algorithm, including calculating support and confidence, generating frequent itemsets, and interpreting association rules. Make sure to show your calculations and provide a comprehensive explanation of each step and the implications of the association rule.

Step 1:

Set the minimum support and confidence thresholds. Let's say we set the minimum support to 40% (meaning an itemset must appear in at least 40% of transactions) and the minimum confidence to 60% (meaning a rule must be correct at least 60% of the time).

Step 2:

Calculate the support for each item (individual items) and identify the frequent items:

Looking at the dataset, we need to calculate the support for each item (individual items) and identify the frequent items based on the minimum support threshold.

bread appears in transactions 1, 2, 4, and 5.
milk appears in transactions 1, 3, and 4.
eggs appear in transaction 1.
diapers appear in transactions 2, 3, 4, and 5.

Calculating the support for each item:

The support for bread is 4/5 or 80%. This means that 80% of the transactions in the dataset contain bread.

The support for milk is 3/5 or 60%. This means that 60% of the transactions in the dataset contain milk.

The support for eggs is 1/5 or 20%. This means that 20% of the transactions in the dataset contain eggs.

The support for diapers is 4/5 or 80%. This means that 80% of the transactions in the dataset contain diapers.

Since the minimum support threshold is set at 40%, the frequent items are those with a support equal to or higher than 40%. In this case, the frequent items are bread, milk, and diapers. Since the minimum count is 2, the item "eggs" does not meet the minimum support threshold and is considered infrequent.

Step 3:

Looking at the dataset we can identify the following pairs:

{bread, milk} appears in transactions 1 and 4.
{bread, diapers} appears in transactions 2, 4, and 5.
{milk, diapers} appears in transactions 3 and 4.

Calculating the support for each pair:

The support for {bread, milk} is 2/5 or 40%. This means that 40% of the transactions in the dataset contain bread and milk together.

The support for {bread, diapers} is 3/5 or 60%. This means that 60% of the transactions in the dataset contain bread and diapers together.

The support for {milk, diapers} is 2/5 or 40%. This means that 40% of the transactions in the dataset contain milk and diapers together.

Step 4:

Generate frequent itemsets of size 3 (triples) using the frequent itemsets of size 2 and calculate their support:

We had previously identified the frequent itemsets of size 2 as follows:

{bread, milk} with a support of 2/5 (40%)
{bread, diapers} with a support of 3/5 (60%)
{milk, diapers} with a support of 2/5 (40%)

To generate frequent itemsets of size 3, we combine these frequent itemsets:

By combining {bread, milk} and {bread, diapers}, we obtain {bread, milk, diapers}. Upon reviewing the dataset:

Transaction 1: {bread, milk, eggs}

Transaction 2: {bread, diapers}

Transaction 3: {milk, diapers}

Transaction 4: {bread, milk, diapers}

Transaction 5: {bread, diapers}

We can see that transaction 4 contains {bread, milk, diapers}, which means the itemset {bread, milk, diapers} appears once in the dataset.

Therefore, the support for the itemset {bread, milk, diapers} is 1/5 or 20%. This means that 20% of the transactions in the dataset contain bread, milk, and diapers together.

Step 5:

Generate association rules from the frequent itemsets and calculate their confidence:

Let's consider the rule {bread, milk} -> {diapers}:

Support({bread, milk, diapers}) = 1/5 (20%)

Confidence({bread, milk} -> {diapers}) = Support({bread, milk, diapers}) / Support({bread, milk})

= 1/5 / 2/5 = 1/2 (50%)

Since the confidence of the rule is lower than the minimum confidence threshold, it is considered an invalid association rule.

An association rule consists of two parts:

an antecedent (left-hand side) and

a consequent (right-hand side).

In this case, the association rule is {bread, milk} -> {diapers}, which means that *if a customer buys bread and milk (antecedent), they are likely to buy diapers as well (consequent).*

To evaluate the rule, we use two measures: support and confidence.

Support measures the proportion of transactions in the dataset that contain a specific itemset. In this case, we need to calculate the support for both the antecedent {bread, milk} and the whole rule {bread, milk, diapers}.

In our example, out of the total of 5 transactions, 2 transactions contain both bread and milk.

So, the support for the itemset {bread, milk} is 2/5 or 40%. This means that 40% of the transactions in the dataset contain bread and milk together.

To calculate the support for the whole rule {bread, milk, diapers}, we need to find the number of transactions that contain all three items: bread, milk, and diapers.

Looking at the dataset again:

Transaction 1: {bread, milk, eggs}

Transaction 2: {bread, diapers}

Transaction 3: {milk, diapers}

Transaction 4: {bread, milk, diapers}

Transaction 5: {bread, diapers}

Out of the 5 transactions, only 1 transaction contains all three items {bread, milk, diapers}.

Therefore, the support for the itemset {bread, milk, diapers} is 1/5 or 20%. This means that 20% of the transactions in the dataset contain bread, milk, and diapers together.

Now, to calculate confidence, we divide the support for the whole rule by the support for the antecedent:

Confidence({bread, milk} -> {diapers}) = Support({bread, milk, diapers}) / Support({bread, milk})
= (1/5) / (2/5)
= 1/2 or 50%

Therefore, the confidence of the rule {bread, milk} -> {diapers} is 50%.

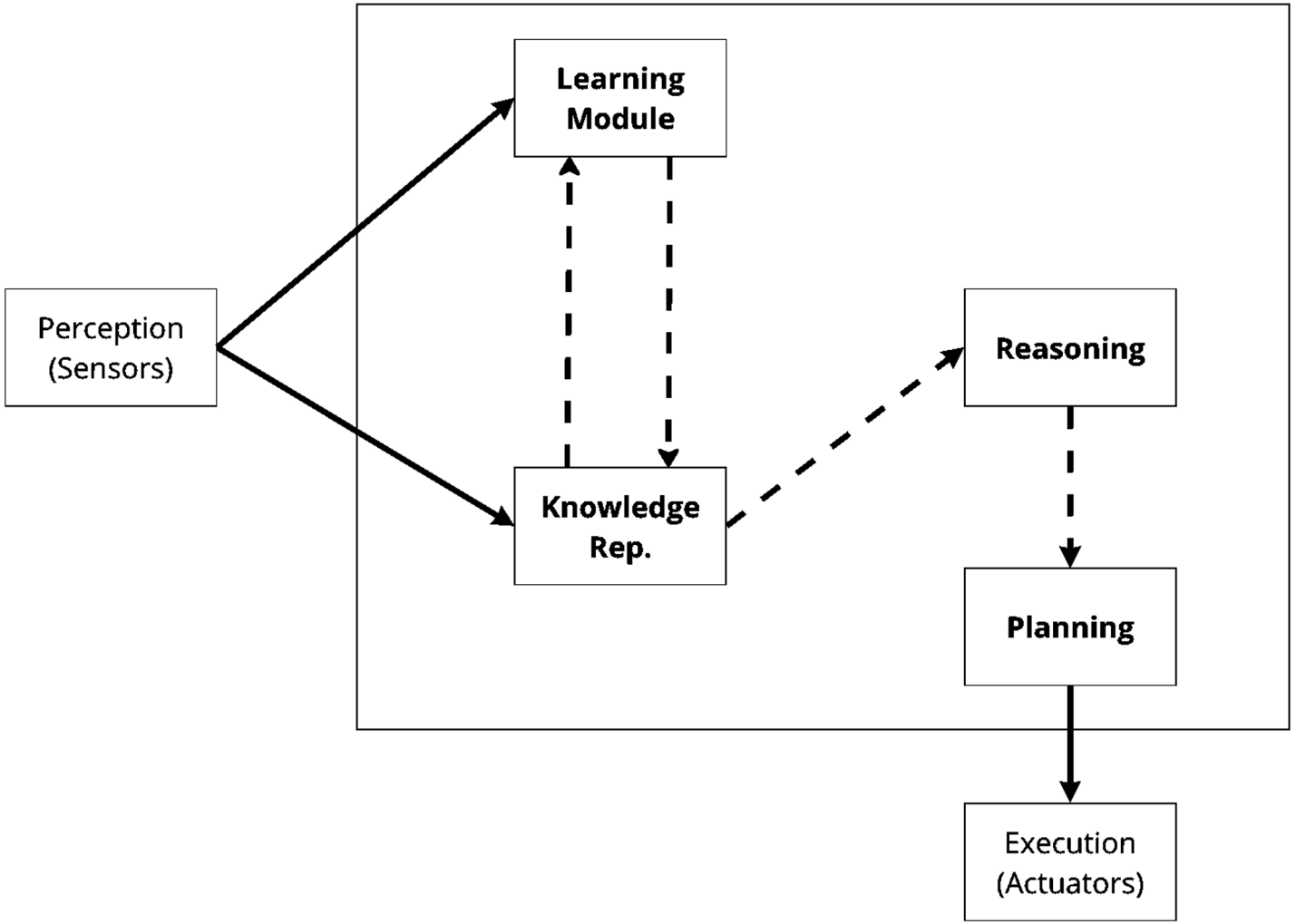
Reinforcement Learning

Reinforcement Learning

- Reinforcement learning is a framework for solving control tasks (also called decision problems) by building agents that learn from the environment by interacting with it through trial and error and receiving rewards (positive or negative) as unique feedback.
- RL is a powerful tool that can be used to solve a wide variety of problems, including game playing, robotics, and finance. Here are some examples of how RL is being used today:
    - Game playing: DeepMind's AlphaGo program used RL to defeat a human Go champion in 2016.
    - Robotics: RL is being used to train robots to perform tasks such as picking and placing objects.
    - Finance: RL is being used to develop trading algorithms that can automatically buy and sell stocks.
- In reinforcement learning, an agent interacts with an environment in a series of discrete time steps. At each time step, the agent observes the current state of the environment, takes an action, and receives feedback in the form of a reward signal and the next state.
- The goal of the agent is to learn a policy, which is a mapping from states to actions, that maximizes the expected cumulative reward over time..

Reinforcement Learning: AI Agent

- *Learning Module*
- *Perception (Sensors)*
- *Knowledge Rep.*
- *Reasoning*
- *Planning*
- *Execution (Actuators)*



Reinforcement Learning

Key concepts in reinforcement learning:

1. Agent: The agent is the entity that is learning to behave in the environment.
2. Environment: The environment is the context in which the agent is operating. It can be physical, such as a game world, or it can be abstract, such as a financial market.
3. State: The state of the environment is a snapshot of all the information that the agent needs to know about the environment in order to make a decision.
4. Action: An action is something that the agent can do to change the state of the environment.
5. Reward: A reward is a signal that tells the agent whether an action was good or bad.
6. Policy: A policy is a rule that tells the agent how to choose actions based on the state of the environment.

Reinforcement Learning: Tasks

Episodic

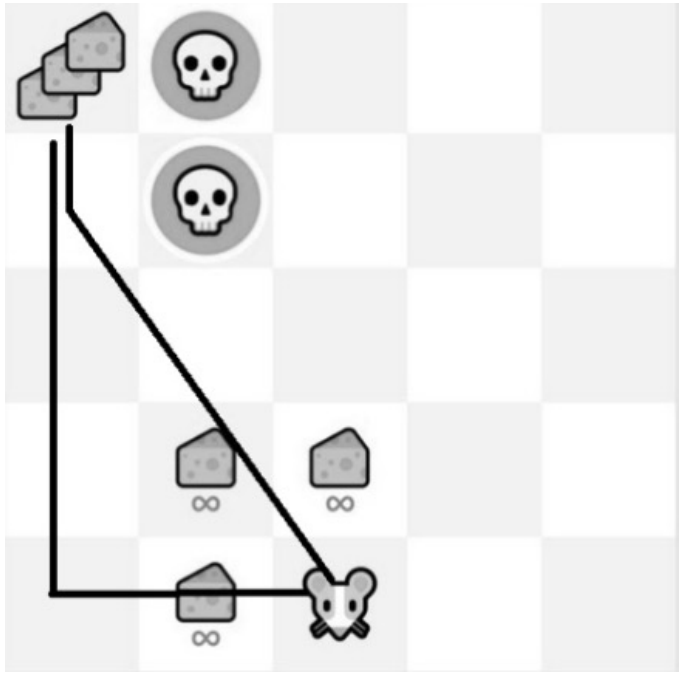
- In this case, we have a starting point and an ending point (a terminal state). This creates an episode: a list of States, Actions, Rewards, and new States.
- For instance, think about a level based game: an episode begin at the launch of a new Level and ends when you fail or you reached the end of the level.

Continuing tasks

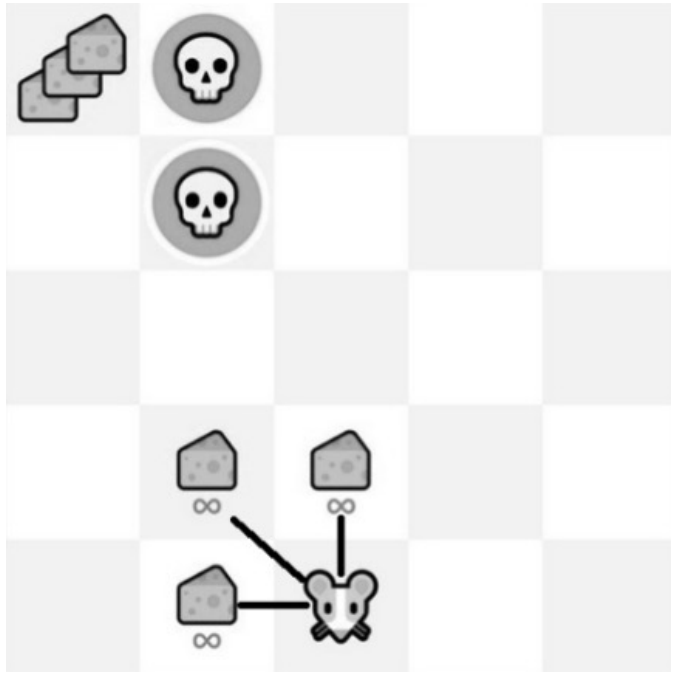
- These are tasks that continue forever (no terminal state). In this case, the agent must learn how to choose the best actions and simultaneously interact with the environment.
- For instance, an agent that does *automated stock trading*. For this task, there is no starting point and terminal state. The agent keeps running until we decide to stop it.

The Exploration/Exploitation trade-off

- Exploration is exploring the environment by trying random actions in order to find more information about the environment.
- Exploitation is exploiting known information to maximize the reward.



Exploration



Exploitation

Reinforcement Learning

- RL algorithms are typically trained using a process called trial and error. The agent starts by exploring the environment and taking random actions. As the agent takes actions, it receives rewards and penalties. Over time, the agent learns to associate certain states with certain actions that lead to rewards. This allows the agent to develop a policy that maximizes its rewards.
- Reinforcement learning has been successfully applied to various domains, including robotics, game playing, autonomous vehicles, recommendation systems, and finance.
- Notable algorithms in this field include

    1. *Q-learning,*
    2. *Deep Q-Networks (DQN),*
    3. *Proximal Policy Optimization (PPO), and*
    4. *Actor-Critic methods.*

Reinforcement Learning: Algorithms

- **Q-Learning:** Q-learning is a model-free, value-based algorithm. It learns the optimal action-value function (Q-function) by iteratively updating the Q-values based on the observed rewards and state transitions. Q-learning is known for its simplicity and has been applied to various tasks, including grid-world navigation and control problems.
- **Deep Q-Networks (DQN):** DQN combines Q-learning with deep neural networks. It uses a neural network to estimate the Q-values, allowing for more complex and high-dimensional state representations. DQN has achieved remarkable success in playing Atari games directly from raw pixels and has been a milestone in the field of reinforcement learning.
- **Policy Gradient Methods:** Policy gradient methods directly learn a parameterized policy that maps states to actions. They optimize the policy by following the gradient of a performance measure, such as the expected cumulative reward. Examples of policy gradient methods include REINFORCE, Proximal Policy Optimization (PPO), and Trust Region Policy Optimization (TRPO).
- **Actor-Critic Methods**: Actor-critic methods combine the benefits of both value-based and policy-based methods. They maintain both a policy (the actor) and a value function (the critic). The actor selects actions based on the policy, while the critic estimates the value function and provides feedback to update the policy. Algorithms like Advantage Actor-Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C) are popular actor-critic methods.
- **Proximal Policy Optimization (PPO):** PPO is a state-of-the-art policy optimization algorithm that addresses some of the limitations of earlier policy gradient methods. It uses a trust region approach to update the policy in a more stable and conservative manner, making it less prone to large policy updates that may lead to instability.

Finding an Optimal Policy

- This Policy is the function we want to learn, our goal is to find the optimal policy that maximizes expected return when the agent acts according to it. We find this through training
- Reinforcement learning algorithms can be categorized into two main types: **value-based methods** and **policy-based methods**.
  - Value-based methods aim to estimate the value function, which represents the expected cumulative reward from a particular state or state-action pair. **(Directly)**
  - Policy-based methods directly learn the policy that maps states to actions without explicitly estimating value functions. **(Indirectly)**
  - There are also hybrid methods that combine both approaches.

Policy-based methods

- Policy-based methods in reinforcement learning are a class of algorithms that directly learn a parameterized policy, which maps states to actions, without explicitly estimating value functions. Instead of optimizing value functions, policy-based methods focus on optimizing the policy itself to maximize the expected cumulative reward.

- In simple terms, we learn a policy function directly. This function will *define a mapping from each state to the best corresponding action*. Alternatively, it could define a probability distribution over the set of possible actions at that state.

Types (*Main*):

- **Deterministic Policy**: A deterministic policy is a mapping from states to actions, where each state is associated with a specific action. It provides a fixed and deterministic action choice for each state. *For example, in a game, a deterministic policy may always select the action "move left" when the agent is in a certain state.*

- **Stochastic Policy:** A stochastic policy assigns a probability distribution over actions for each state. Instead of selecting a single action, the policy outputs a probability distribution that specifies the likelihood of taking each action. This allows for exploration and introduces randomness into the decision-making process. Stochastic policies are often used to balance exploration and exploitation. *For example, in a game, a stochastic policy may assign higher probabilities to actions that have been successful in the past.*

Policy-based methods

Types

- **Deterministic Policy**: action = policy(state)
- **Stochastic Policy:** policy(actions | state) = probability distribution over the set of actions given the current state

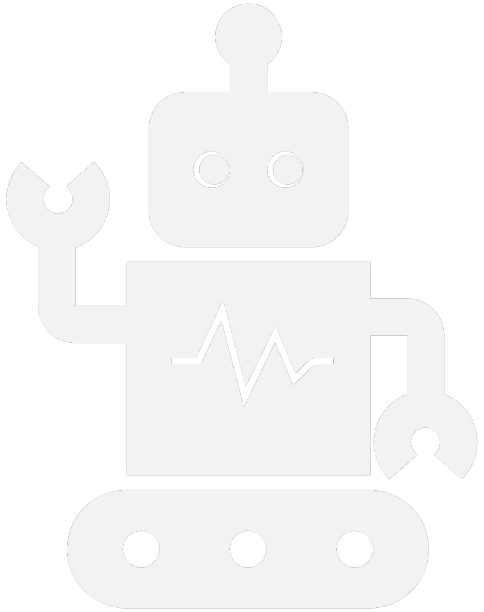
Advantages of Policy-Based Methods:

1. **Flexibility:** Policy-based methods can learn stochastic policies that explore different actions, which can be beneficial in complex and uncertain environments.
2. **Continuous Action Spaces:** Policy-based methods can handle continuous action spaces more naturally compared to value-based methods, as they directly output action probabilities.
3. **High-Dimensional State Spaces:** By using function approximators, such as neural networks, policy-based methods can handle high-dimensional state representations, such as images.
4. **Convergence:** Policy-based methods can converge to locally optimal policies and can be more stable compared to value-based methods, especially in the presence of high-dimensional or noisy state spaces.

Value-based methods

- These focus on estimating and optimizing value functions, which assign values to states or state-action pairs. These methods aim to find the optimal value function that maximizes the expected cumulative reward.
- The idea is that we learn a value function that maps a state to the expected value of being at that state. The value of a state is the expected discounted return the agent can get if it starts in that state, and then acts according to our policy.
- There are two main types of value-based methods: state-value methods and action-value methods.
    1. State-value methods learn a value function that maps states to their expected return.
    2. Action-value methods learn a value function that maps state-action pairs to their expected return.
- For the state-value function, we calculate the value of a state.
- For the action-value function, we calculate the value of the state-action pair hence the value of taking that action at that state.
- The most well-known value-based method is Q-learning. Q-learning is an off-policy temporal difference (TD) learning algorithm that learns an action-value function.

Lab



Q-Learning Algorithm

Q-Learning

- Q-learning is a simple and popular algorithm in the field of reinforcement learning, which is a type of machine learning. It allows an agent to learn optimal actions to take in a given environment by trial and error.
- At its core, Q-learning uses a table called a Q-table to store and update the estimated values of taking different actions in different states of the environment. The Q-values represent the expected cumulative reward the agent will receive if it takes a particular action in a specific state.
- The Q-learning algorithm is guaranteed to converge to the optimal Q-values if the environment is deterministic and the discount factor is less than 1. However, in practice, it may take a long time for the algorithm to converge, especially for large and complex environments.
- Q-learning has been used to solve a wide variety of reinforcement learning problems, including playing games, controlling robots, and optimizing financial trading strategies.

Q-Learning Algorithm

1. Initialize the Q-values to arbitrary values.

2. Repeat:

    1. Take an action in the current state according to a policy that is greedy with respect to the current Q-values.

    2. Observe the next state and reward.

    3. Update the Q-values for the current state and action using the following formula:

$$Q(s, a) = (1 - alpha) * Q(s, a) + alpha * (reward + gamma * max_a Q(next\_state, a))$$

Where

- alpha is the learning rate
- gamma is the discount factor
- reward is the immediate reward received for taking action a in state s
- next_state is the state that the agent transitioned to after taking action a
- max_a Q(next_state, a) is the maximum Q-value for all actions in the next state

Q-Learning

Advantages:

- It is a model-free algorithm, which means that it does not require knowledge of the environment dynamics.
- It can handle problems with stochastic transitions and rewards.

Disadvantages:

- It can be slow to converge, especially for large and complex environments.
- It can be sensitive to the choice of hyperparameters, such as the learning rate and the discount factor.
- It can suffer from overfitting, which means that it may learn the specific training data too well and not generalize well to new data.

Q-Learning

Epsilon-Greedy Strategy

- Epsilon-greedy is an exploration strategy used in Q-learning. The agent selects actions based on a balance between exploration and exploitation. It works as follows: when the agent needs to make a decision, it randomly chooses a number between 0 and 1.
- If the random number is less than a pre-defined epsilon value, the agent explores by selecting a random action. If the random number is greater than or equal to epsilon, the agent exploits its learned knowledge by selecting the action with the highest expected reward.
- By adjusting the epsilon value, the agent can control the level of exploration versus exploitation, gradually refining its decision-making process.

Q-Learning

The Discount Factor

- The discount factor (gamma) is a parameter used in Q-learning to balance immediate rewards and future rewards. It determines how much importance the agent places on future rewards compared to immediate rewards.
- A value between 0 and 1 is chosen for gamma. If gamma is closer to 0, the agent heavily discounts future rewards, prioritizing immediate gains. If gamma is closer to 1, the agent values future rewards more, considering the cumulative reward it can achieve in the long run.
- The discount factor allows the agent to make decisions that strike a balance between short-term gains and long-term benefits, adapting its strategy based on the importance of immediate versus future rewards.

Q-Learning

The Learning Rate

- The learning rate (α) is a parameter that determines the extent to which new information overrides or influences existing Q-values. It controls the rate at which the Q-values are updated during the learning process.
- A high learning rate means that the new information has a strong influence on updating the Q-value. In this case, the agent quickly adapts to the received reward and adjusts the Q-value of the state-action pair accordingly. However, a high learning rate can also lead to instability or overfitting to the observed rewards, especially if the environment is stochastic or noisy.
- A low learning rate means that the new information has a smaller impact on the Q-value. The agent learns more slowly and requires more experiences to update the Q-values effectively. However, a low learning rate provides more stability and prevents the agent from overreacting to noisy rewards or making drastic changes based on limited experiences.

Learning rate = 0.1

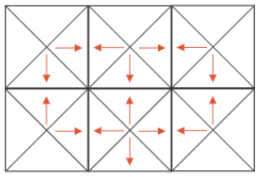
Epsilon = 1 (this decays each timestep i.e. reduces slowly towards the value 0)

Discount Factor = 0.99

Environment

S (*)	1	0
0	-10	10

Initialise Q-Table (Q-Values) && 4 unique actions (up, right, down, left)



	UP	RIGHT	DOWN	LEFT
S	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**
-10	**0**	**0**	**0**	**0**
10	**0**	**0**	**0**	**0**

For now, **our Q-table is useless**; we need **to train our Q-function using the Q-Learning algorithm**

Training Timestep 1:

Choose an action using the Epsilon Greedy Strategy (Epsilon = 1, it will be decayed with time i.e. reduced) Because epsilon is big (= 1.0), We can take a random action.

	*	0
0	-10	10

Moving to the right gets a reward of 1

We then need to update our q-value for this State - Action pair i.e. moving to the right while at the starting state.

	UP	RIGHT	DOWN	LEFT
S	**0**	**0**	**0**	**0**

To make this update, we use the Q-learning formula

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation | Former Q-value estimation | Learning Rate | Immediate Reward | Discounted Estimate optimal Q-value of next state | Former Q-value estimation

TD Target

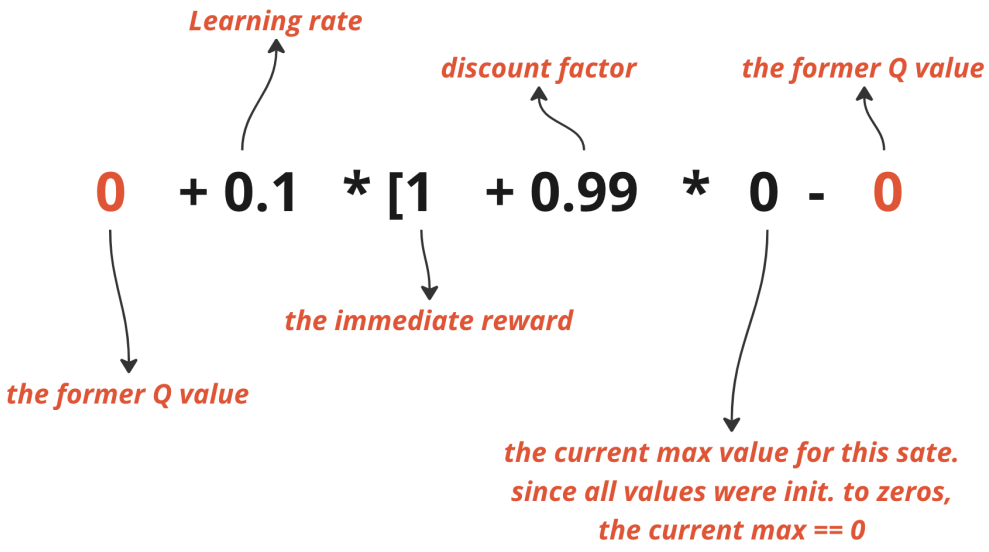
TD Error

Therefore:

- Q(S, Right) is given by: **0 + 0.1 * [1 + 0.99 * 0 - 0] = 0.1**

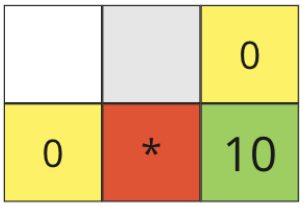
	UP	RIGHT	DOWN	LEFT
S	0	0.1	0	0

Eq. from above



Training Timestep 2:

We first decay the epsilon slightly i.e. from 1 to 0.99
Because epsilon is still high (= .99), We can take another random action. e.g moving **down**



Moving to the down gets a reward of -10

We then need to update our q-value for this State - Action pair i.e. moving to the right while at the starting state.

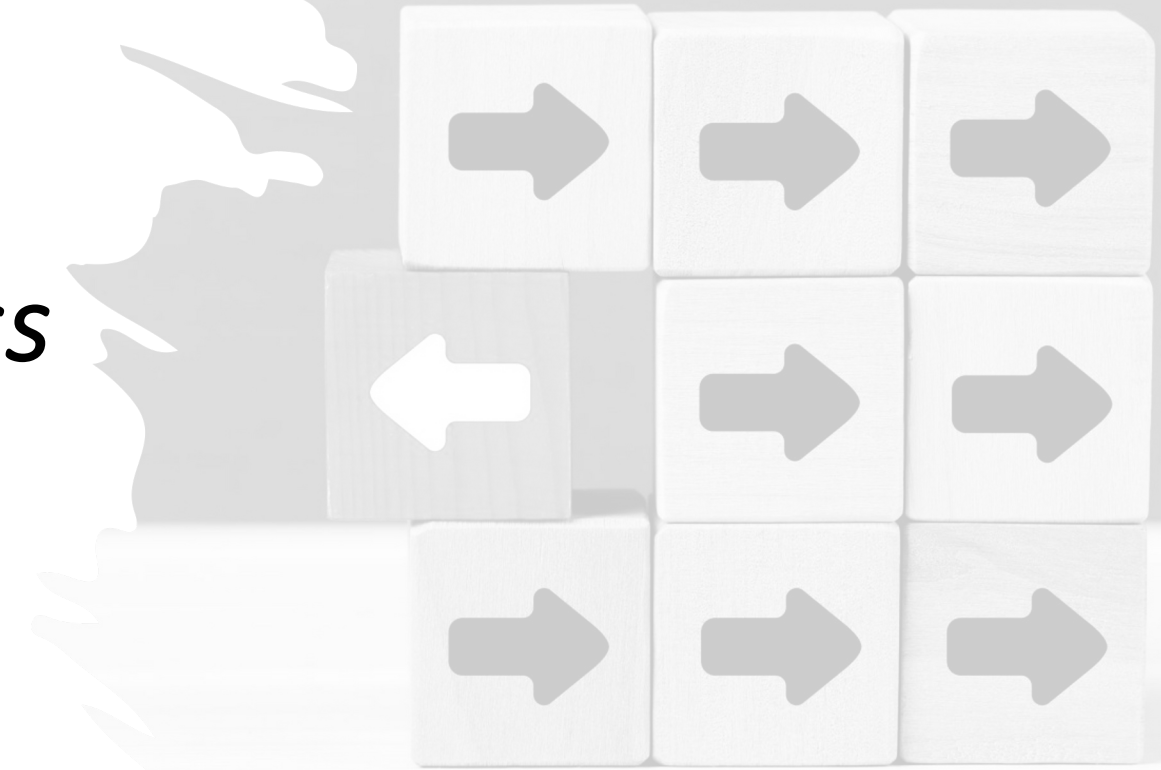
	UP	RIGHT	DOWN	LEFT
S	**0**	**0.1**	**0**	**0**
1	**0**	**0**	**0**	**0**

Therefore:

- Q(1, Down) is given by: **0 + 0.1 * [-10 + 0.99 * 0 - 0] = -1**

	UP	RIGHT	DOWN	LEFT
S	**0**	**0.1**	**0**	**0**
1	**0**	**0**	**-1**	**0**

Markov Decision Process (MDP)



Markov Decision Process (MDP)

- A Markov decision process (MDP) is a mathematical model that describes a system where an agent can take actions to move from one state to another, and each state has a probability of generating a reward.
- A simple example of an MDP is a game of Tic-Tac-Toe. The agent (the player) can take actions by placing an X or an O in one of the nine squares on the board. Each state of the game is a complete configuration of the board, and each state has a probability of generating a reward (winning, losing, or drawing).
- MDPs can be used to solve problems by finding a policy, which is a mapping from states to actions. The optimal policy is the policy that maximizes the expected reward over time.
- There are a number of different algorithms that can be used to find the optimal policy for an MDP. One common approach is value iteration, which iteratively updates the value of each state until the values converge.
- MDPs are a powerful tool for modelling and solving decision-making problems. They are used in a variety of applications, and they continue to be a topic of active research.

Markov Decision Process (MDP)

Components

- States (S): A set of possible states that the agent can be in.
- Actions (A): A set of possible actions that the agent can take.
- Transition Probabilities (P): The probabilities of transitioning from one state to another when an action is taken.
- Rewards (R): The immediate rewards associated with state-action pairs.
- Discount Factor (γ): A value between 0 and 1 that determines the importance of future rewards.

Markov Property

The MDP assumes the Markov property, which states that the future state depends only on the current state and action, and not on the past history of states and actions.

Markov Decision Process (MDP)

Policy (π)

A policy is a rule or strategy that determines the agent's action selection in each state. It maps states to actions. A policy can be deterministic, where it directly assigns a specific action to each state, or stochastic, where it assigns probabilities to different actions in each state.

Value Function

- State Value Function (V): The expected cumulative reward starting from a particular state and following a given policy.
- Action Value Function (Q): The expected cumulative reward starting from a particular state, taking a specific action, and following a given policy.

Bellman Equations

- Bellman Expectation Equation for V: The value of a state is the immediate reward plus the discounted value of the next state, according to the policy.
- Bellman Expectation Equation for Q: The value of a state-action pair is the immediate reward plus the discounted value of the next state-action pair, according to the policy.

Markov Decision Process (MDP)

Markov Decision Processes (MDPs) have numerous applications across various domains. Some notable use cases of MDPs include:

- Robotics: MDPs can be used to model and solve decision-making problems in robotics, such as robot navigation, path planning, and task scheduling. MDPs enable robots to make optimal decisions while considering uncertain environments and long-term goals.
- Autonomous Vehicles: MDPs are applied in autonomous vehicle systems to determine optimal driving strategies, including lane changing, speed control, and trajectory planning. MDPs consider factors like traffic conditions, road constraints, and safety requirements to make informed decisions.
- Resource Management: MDPs are utilized in resource allocation and management problems. For example, in energy management systems, MDPs can optimize power generation and distribution by considering factors like energy demand, renewable sources, and cost constraints.
- Healthcare: MDPs find applications in healthcare settings, such as personalized treatment planning and resource allocation in hospitals. MDPs can optimize treatment decisions, patient scheduling, and resource allocation to improve patient outcomes and operational efficiency.

Markov Decision Process (MDP)

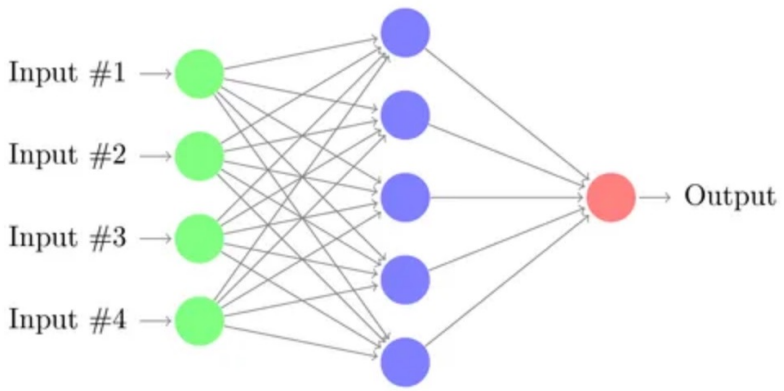
- Finance: MDPs are employed in financial domains for portfolio optimization, trading strategies, and risk management. MDPs help in making investment decisions by considering market dynamics, risk tolerance, and expected returns.
- Game AI: MDPs are used in game development for creating intelligent game agents. MDPs model the game environment and enable agents to make optimal decisions in games like chess, poker, and video games.
- Control Systems: MDPs play a significant role in control systems, such as industrial automation and process control. MDPs help in optimizing control policies, determining setpoints, and dealing with uncertain and stochastic dynamics.
- Natural Language Processing: MDPs are employed in natural language processing tasks, such as dialogue systems and machine translation. MDPs assist in generating appropriate responses and optimizing dialogue interactions based on the context.

Convolutional Neural Networks

CNNs

The Problem with Traditional Neural Networks

These are built based on the human brain, whereby neurons are stimulated by connected nodes and are only activated when a certain threshold value is reached.



Since MLPs use one perceptron for each input (e.g. pixel in an image, multiplied by 3 in RGB case). The amount of weights rapidly becomes unmanageable for large images. For a 224 x 224 pixel image with 3 color channels there are around 150,000 weights that must be trained! As a result, difficulties arise whilst training and overfitting can occur.

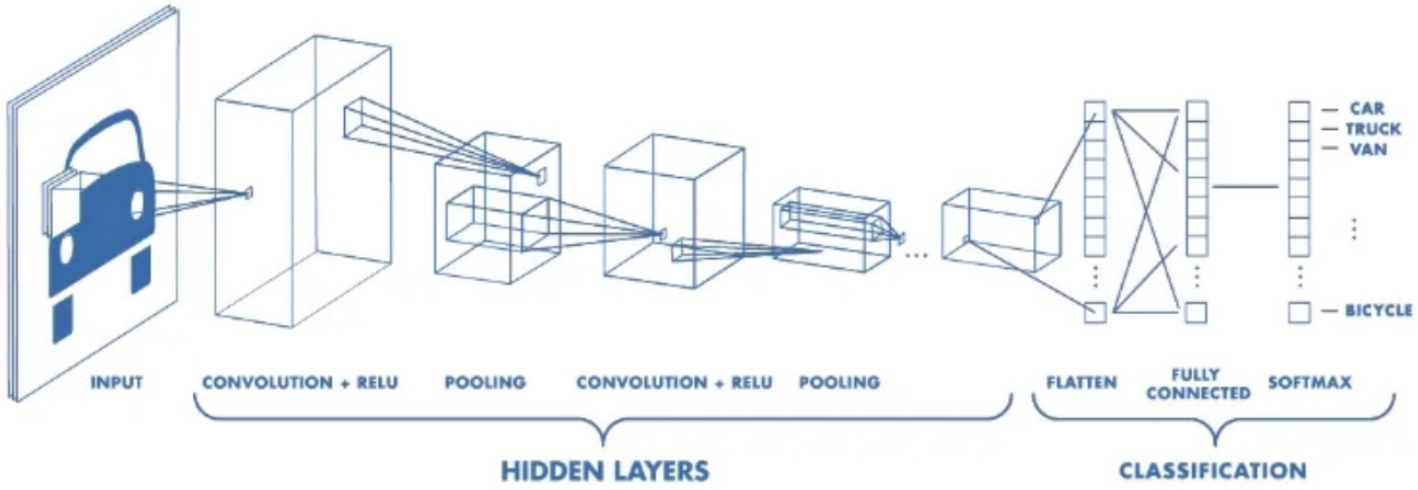
Another common problem is that MLPs react differently to an input (images) and its shifted version — they are not translation invariant. For example, if a picture of a cat appears in the top left of the image in one picture and the bottom right of another picture, the MLP will try to correct itself and assume that a cat will always appear in this section of the image.

CNNs

- Multilayer perceptrons (MLPs) are not well-suited for image processing because they lose spatial information when the image is flattened into a vector. This is a problem because the spatial relationships between pixels are important for defining the features of an image.
- For example, if an MLP is trained on a dataset of cat images, it will learn to identify the features of a cat regardless of its position in the image. However, if the cat is in a different position in the test image, the MLP may not be able to recognize it. This is because the MLP has not learned to take into account the spatial relationships between the pixels.
- A better approach for image processing is to use a convolutional neural network (CNN). CNNs are able to learn the spatial relationships between pixels, which makes them more robust to changes in the position of objects in an image.
- For example, a CNN that is trained on a dataset of cat images will be able to recognize a cat in a test image, regardless of its position in the image.

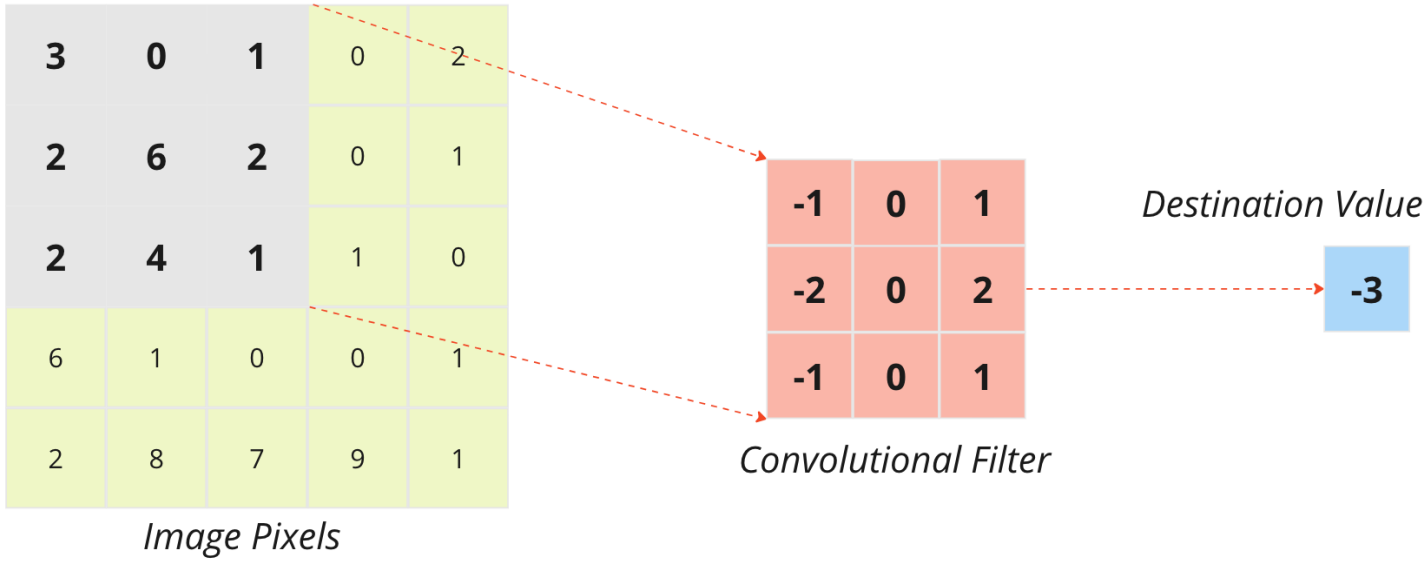
CNNs

- A convolutional neural network (CNN) is a type of artificial neural network that is specifically designed for processing data that has a grid-like structure, such as images. CNNs are able to learn to identify patterns in images by applying a series of convolution operations to the input data.
- Convolution is a mathematical operation that takes two functions and produces a third function that is a combination of the two. In the context of CNNs, the two functions are the input image and a filter. The filter is a small matrix of weights that is used to scan the input image. The output of the convolution operation is a new image that contains the features that were detected by the filter.
- CNNs are typically composed of several layers of convolution operations, followed by pooling layers and fully-connected layers. The pooling layers reduce the size of the output from the convolution layers, which helps to reduce the computational complexity of the network. The fully-connected layers then combine the output from the pooling layers to produce a final output that represents the class of the input image.



The Convolution Process

- Imagine you have an image, which is a grid of pixels arranged in rows and columns. Each pixel represents the intensity or color of a specific point in the image.
- Now, the convolution operation in CNN involves a small filter or kernel sliding across the input image. The filter is a small matrix of numbers that serves as a feature detector. It can have different dimensions, such as 3x3 or 5x5.
- As the filter slides across the image, it performs an element-wise multiplication between the values of the filter and the corresponding pixels it is currently positioned on. These multiplications are summed up, resulting in a single value.



$$(-1 * 3) + (0 * 0) + (1 * 1) + (-2 * 2) + (0 * 6) + (2 * 2) + (-1 * 2) + (0 * 4) + (1 * 1)$$

$$(-2) + (0) + (-1)$$

The Convolution Process

- The values for the filter in a Convolutional Neural Network (CNN) are learned through the training process. During training, the network adjusts the filter's values (also known as the filter's weights or parameters) based on the given task and the available labeled data.
- In the beginning, the filter's values are usually initialized randomly or with small random values close to zero. As the training progresses, the network updates these weights to minimize the difference between the predicted output and the true output.
- The update of the filter's weights is performed using a technique called backpropagation. Backpropagation calculates the gradients of the loss function with respect to the filter's weights, indicating the direction and magnitude of the weight adjustments needed to improve the network's performance. These gradients are propagated backward through the network, and the weights are updated accordingly using an optimization algorithm, such as stochastic gradient descent (SGD) or Adam.
- Through the iterative training process, the network gradually learns to adjust the filter's values to capture relevant features from the input data. The network discovers which filter values are most effective in detecting specific patterns or features that are important for the given task.
- In a Convolutional Neural Network (CNN), the training process is essentially focused on improving the feature extraction from images. CNNs are designed to automatically learn and extract relevant features directly from the input data, which is particularly beneficial for image-related tasks.

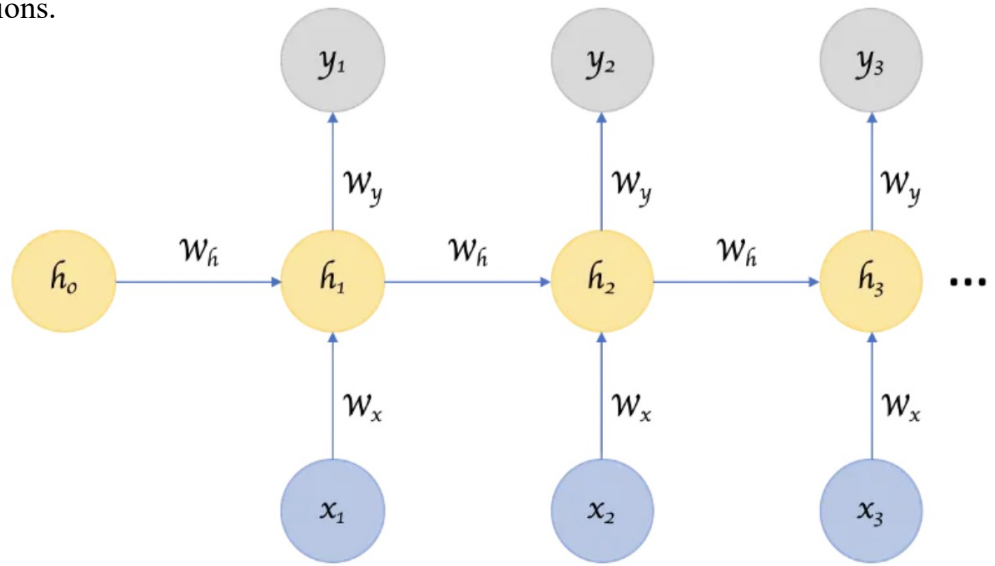
Recurrent Neural Networks

(RNNs)

Recurrent Neural Networks

- RNNs can be thought of as a series of connected neural networks, each of which is responsible for processing one step of the input sequence. The output of each network is then fed as input to the next network in the series. *This allows RNNs to learn the relationships between different steps in a sequence, and to predict future steps in the sequence*. In traditional neural networks, all the inputs and outputs are independent of each other, but this is not a good idea if we want to predict the next word in a sentence

- Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. To evaluate a new input x(t) at time t, we combine x(t) with the previous hidden state h(t-1) to generate new predictions.



Recurrent Neural Networks

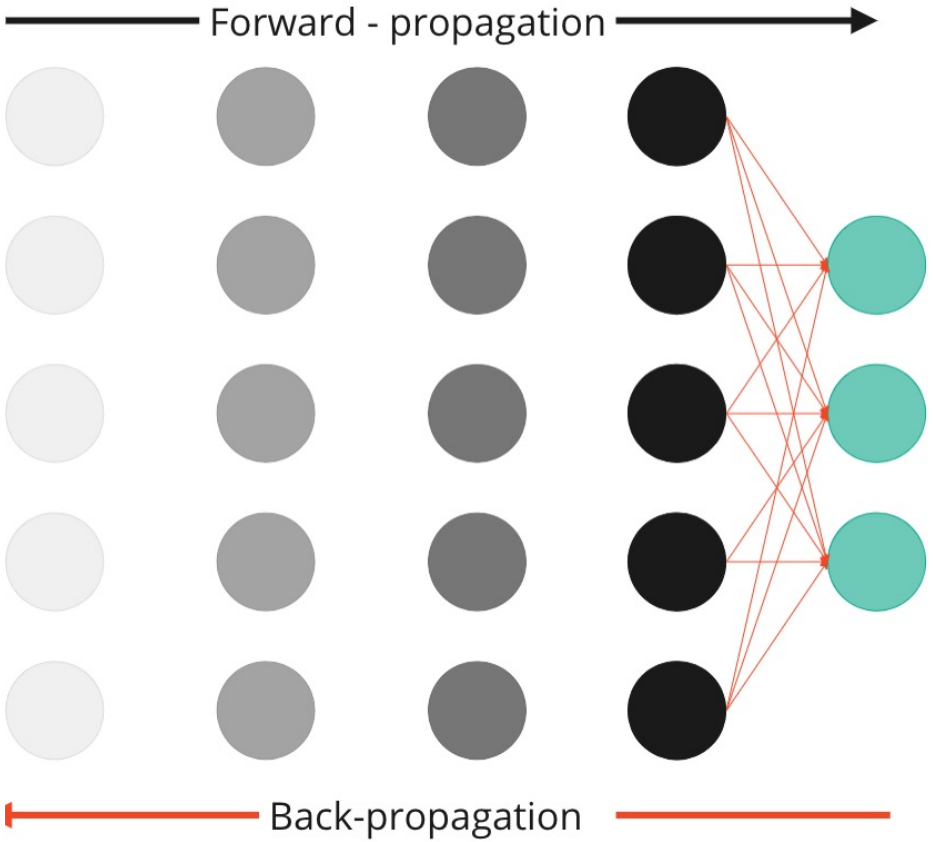
- Let's say we want to train an RNN to predict the next word in a sentence. The RNN would be given the current word as input, and it would output a prediction for the next word.
- However, the RNN would also keep track of the previous words that it had seen, and it would use this information to make its prediction. This allows the RNN to learn long-term dependencies between words, which is essential for tasks like machine translation and text generation.
- Long-term dependencies in RNNs can be a good thing, but they can also be a problem.
    - *On the one hand, they allow RNNs to learn relationships between data points that are far apart in time. This can be very useful for tasks like machine translation and text generation, where it is important to understand the context of a word or phrase.*
    - *On the other hand, long-term dependencies can make it difficult for RNNs to learn from short sequences of data. This is because the gradients that are used to update the RNN's parameters can become very small over long sequences of data.*
- There are a number of techniques that can be used to address the problem of long-term dependencies in RNNs.
- One common technique is to use a regularization technique called dropout. *Dropout randomly drops out units in the RNN during training*. This helps to prevent the RNN from relying too heavily on any particular unit, and it can help to prevent the gradients from becoming too small.

Recurrent Neural Networks

- Another technique is to use *Gated Recurrent Unit* (GRUs) and *Long Short-Term Memory* (LSTMs).
- GRUs and LSTMs *are both gated RNNs*, which means that they have gates that control the flow of information through the network. These gates allow the RNN to decide which information is important to remember for long periods of time, and which information can be discarded.
- GRUs and LSTMs have been shown to be very effective at learning long-term dependencies. They have been used successfully in a wide variety of applications, including machine translation, text generation, and speech recognition.

Feature	GRU	LSTM
Number of gates	2	3
Gate type	Reset gate, update gate	Forget gate, input gate, output gate
Complexity	Less complex	More complex
Accuracy	Similar	LSTM is generally more accurate
Speed	Faster	Slower

Vanishing Gradient Problem



- The vanishing gradient problem is a phenomenon that occurs during the training of deep recurrent neural networks (RNNs), where *the gradients that are used to update the network become extremely small or "vanish" as they are backpropagated from the output layers to the earlier layers*.
- This can make it difficult to train the network, as the gradients become too small to effectively update the weights.
- The vanishing gradient problem is caused by the fact that the gradients are multiplied together as they are backpropagated through the network. If the gradients are small, then they will be multiplied together to become even smaller.
- This can happen if the activation functions in the network are sigmoid or tanh functions, as these functions have a derivative that is always less than 1.

Vanishing Gradient Problem

Some of the solutions to the vanishing gradient problem in RNNs include:

1. Use a different activation function. The sigmoid and tanh functions are not the only activation functions that can be used in neural networks. Other activation functions, such as the ReLU function, have a derivative that is always greater than 0. This means that the gradients will not vanish as they are backpropagated through the network.

2. Use a regularization technique. Regularization techniques can help to prevent the network from becoming too complex, which can help to prevent the gradients from becoming too small. Dropout is a common regularization technique that can be used to address the vanishing gradient problem.

3. Use a gated RNN. Gated RNNs have gates that control the flow of information through the network. This allows the RNN to decide which information is important to remember for long periods of time, and which information can be discarded. Gated RNNs such as LSTMs and GRUs have been shown to be very effective at addressing the vanishing gradient problem.

4. Use truncated backpropagation through time (BPTT). Truncated BPTT is a technique that limits the number of time steps that are backpropagated through the network. This can help to prevent the gradients from becoming too small.

More Aspects on Training Neural Networks

Tuning Neural Networks

- In machine learning, we need to differentiate between parameters and hyperparameters. A learning algorithm learns or estimates model parameters for the given data set, then continues updating these values as it continues to learn.
- After learning is complete, these parameters become part of the model. For example, each weight and bias in a neural network is a parameter.
- Hyperparameters, on the other hand, are specific to the algorithm itself, so we can't calculate their values from the data. We use hyperparameters to calculate the model parameters.
- Different hyperparameter values produce different model parameter values for a given data set.
- Hyperparameter tuning consists of finding a set of optimal hyperparameter values for a learning algorithm while applying this optimized algorithm to any data set. That combination of hyperparameters maximizes the model's performance, minimizing a predefined loss function to produce better results with fewer errors.
- Note that the learning algorithm optimizes the loss based on the input data and tries to find an optimal solution within the given setting.

Tuning Neural Networks

Some important hyperparameters that require tuning in neural networks are:

- Number of hidden layers: It’s a trade-off between keeping our neural network as simple as possible (fast and generalized) and classifying our input data correctly. We can start with values of four to six and check our data’s prediction accuracy when we increase or decrease this hyperparameter.
- Number of nodes/neurons per layer: More isn't always better when determining how many neurons to use per layer. Increasing neuron count can help, up to a point. But layers that are too wide may memorize the training dataset, causing the network to be less accurate on new data.
- Learning rate: Model parameters are adjusted iteratively — and the learning rate controls the size of the adjustment at each step. The lower the learning rate, the lower the changes to parameter estimates are. This means that it takes a longer time (and more data) to fit the model — but it also means that it is more likely that we actually find the minimum loss.
- Momentum: Momentum helps us avoid falling into local minima by resisting rapid changes to parameter values. It encourages parameters to keep changing in the direction they were already changing, which helps prevent zig-zagging on every iteration. Aim to start with low momentum values and adjust upward as needed.

Dropouts in Neural Networks

- Dropout is a regularization technique commonly used in neural networks to prevent overfitting. It was introduced by Srivastava et al. in 2014 and has since become a popular and effective technique in deep learning.

How does Dropout work?

- During training, at each iteration, dropout is applied independently to each unit with a certain probability. This probability is typically set between 0.2 and 0.5, but the actual value depends on the problem and the network architecture.
- In each iteration, different units are dropped out, creating a different subnetwork. The idea behind dropout is to prevent units from relying too much on each other and promote the learning of more robust and generalized features.
- During testing or inference, dropout is usually turned off or modified to account for the fact that all units are active. This is done by scaling the output values of the units by the dropout probability.
- For example, if a unit has a dropout probability of 0.5, its output value during testing is multiplied by 0.5 to account for the fact that, on average, only half of the units were active during training.

Dropouts in Neural Networks

Benefits of Dropout:

1. Regularization: Dropout acts as a regularization technique, helping to reduce overfitting by preventing complex co-adaptations between units. It prevents units from relying too heavily on each other and encourages the network to learn more diverse and independent features.

2. Ensemble learning: Dropout can be seen as a form of ensemble learning, where multiple subnetworks are trained with different units dropped out. By averaging the predictions of these subnetworks during testing, dropout can improve the model's overall performance and robustness.

3. Reduces complex co-adaptations: Dropout discourages units from relying too much on specific input features or other units. This helps to break up complex co-adaptations, making the network more robust to changes in the input and reducing the risk of overfitting.

4. Computational efficiency: Dropout is computationally efficient since the dropout masks can be easily applied during both forward and backward passes without introducing significant computational overhead.

Dropouts in Neural Networks

Considerations when using Dropout

1. Dropout rate: The dropout rate determines the fraction of units that are dropped out during training. A dropout rate that is too high may result in underfitting, as the network may not have enough units to learn complex patterns. On the other hand, a dropout rate that is too low may not provide enough regularization, and the network may still be prone to overfitting.

2. Layer placement: Dropout can be applied to input layers, hidden layers, or both. Generally, applying dropout to hidden layers tends to be more effective in preventing overfitting, as it introduces noise at multiple stages of the network and allows for more diverse representations to be learned.

3. Impact on training time: Dropout can slow down the training process since it requires multiple forward and backward passes for each training iteration. However, this additional time cost is often offset by the regularization benefits of dropout.

4. Combining with other regularization techniques: Dropout can be combined with other regularization techniques, such as L1 or L2 regularization, to further improve model performance. These techniques can complement each other and provide additional regularization benefits.

Early Stopping Neural Networks

- Early stopping is a technique used in machine learning, including neural networks, to prevent overfitting and improve the generalization performance of the model. It involves monitoring the performance of the model during training and stopping the training process early if certain criteria are met.
- Instead of training the model for a fixed number of epochs, early stopping monitors the performance of the model on a separate validation set and stops training when the validation performance starts to degrade.
- During training, the performance of the model is evaluated periodically on the validation set. Typically, this evaluation is done after each training epoch, but it can also be performed at regular intervals. The performance metric used for monitoring can be the validation loss or any other relevant metric, such as accuracy or F1 score, depending on the problem.
- Early stopping involves defining criteria for when to stop the training process. The criteria are based on observing the validation performance over several iterations. The most common approach is to track the validation loss. If the validation loss stops improving or starts to increase consistently for a certain number of epochs, the training is stopped early.
- When early stopping is triggered, the training process stops, and the model at that point is considered the best model. The best model is determined based on the performance on the validation set. This model can then be used for inference or further evaluation on unseen data.

Early Stopping Neural Networks

Benefits of Early Stopping:

1. Prevents overfitting: Early stopping helps prevent the model from becoming too specialized in the training data, leading to better generalization on unseen examples.

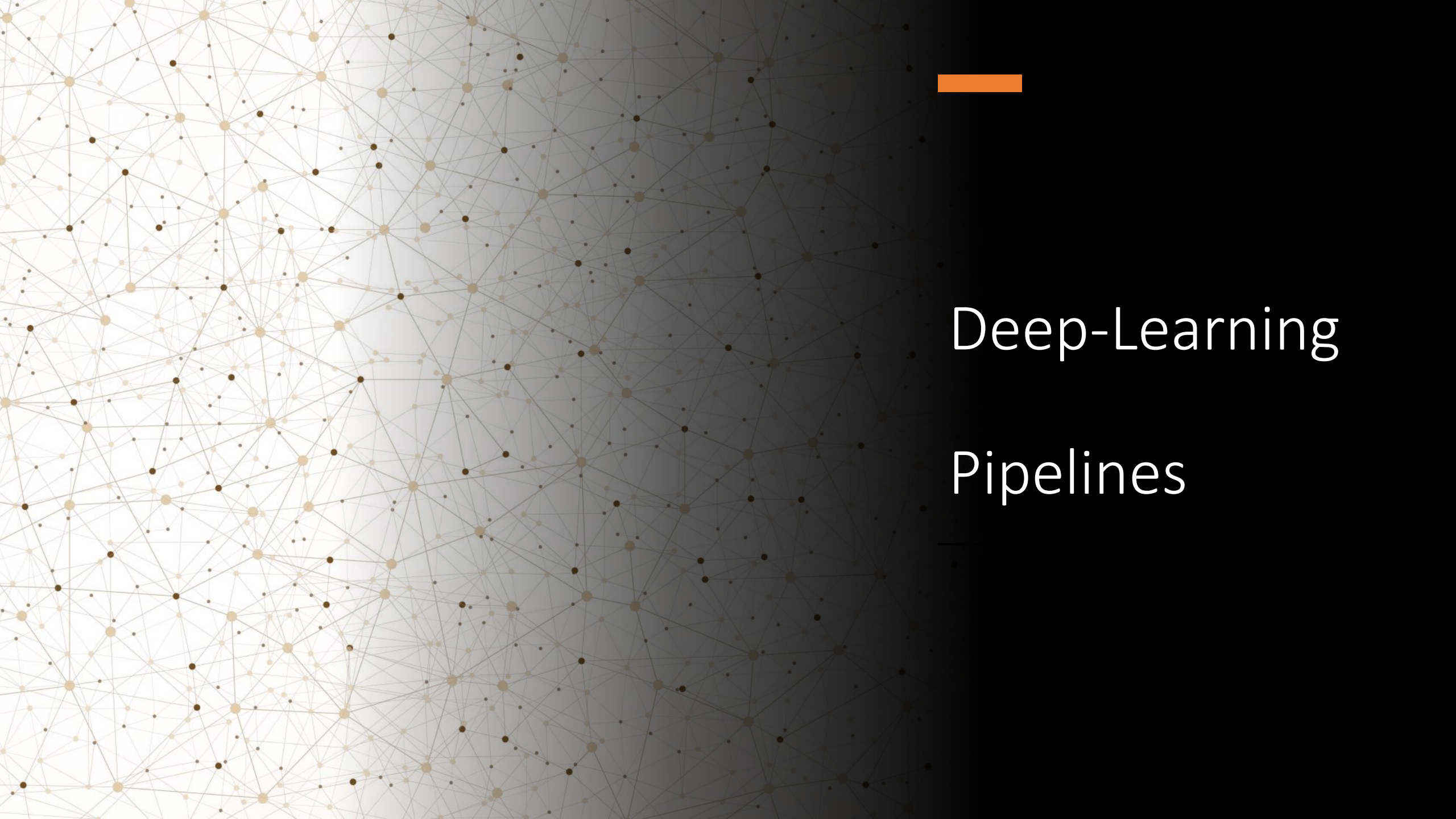
2. Saves computation time: Training a neural network can be computationally expensive, especially for large models and datasets. Early stopping avoids unnecessary training iterations, saving computational resources.

3. Simplicity and interpretability: Early stopping is a simple and interpretable technique that doesn't require additional hyperparameters or complex regularization techniques.

Considerations:

1. Early stopping should be used in combination with other regularization techniques, such as dropout or weight decay, to maximize performance and prevent overfitting.

2. The choice of the number of epochs to wait before stopping can vary depending on the problem and dataset. It requires experimentation and validation set monitoring to find the optimal stopping point.



Deep-Learning Pipelines

What is Deep Learning

- Deep learning is a subfield of machine learning that focuses on developing algorithms and models inspired by the structure and function of the human brain, particularly artificial neural networks.
- It involves training neural networks with multiple layers (hence the term "deep") to learn patterns and representations from large amounts of data.
- At its core, deep learning leverages artificial neural networks, which are computational models composed of interconnected nodes, or artificial neurons, organized into layers. Each neuron receives input, performs a computation, and produces an output signal that is passed to the next layer.
- The connections between neurons have associated weights that are adjusted during the training process to optimize the network's performance.
- Deep learning has gained significant attention and popularity in recent years due to its ability to automatically learn hierarchical representations from raw data.
- This eliminates the need for manual feature engineering, where domain experts design and extract relevant features for a specific task. Deep learning models can learn directly from raw data, such as images, text, audio, or sensor data, by automatically extracting meaningful features and patterns at different levels of abstraction.

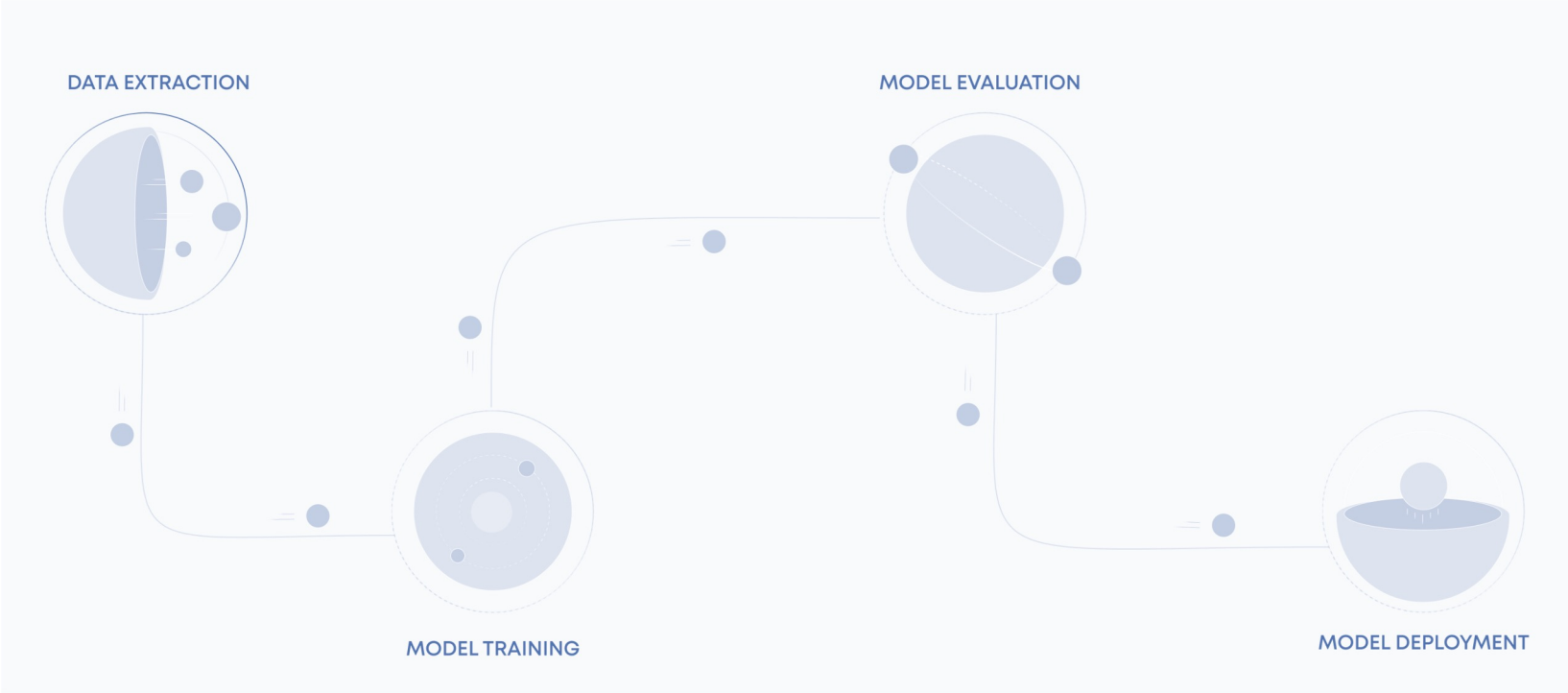
What is Deep Learning

- One of the key strengths of deep learning is its ability to handle high-dimensional data and learn complex relationships. Deep neural networks can capture intricate patterns and dependencies in the data, enabling them to excel in tasks such as image and speech recognition, natural language processing, recommender systems, and many other domains.
- Here, models are typically trained using large datasets and powerful computational resources. Training involves presenting the network with labelled examples, allowing it to adjust its weights and improve its predictions through an optimization process called backpropagation.
- The objective is to minimize the difference between the predicted outputs of the network and the true labels, often using techniques like gradient descent and stochastic gradient descent.
- While deep learning has achieved remarkable success in various domains, it also comes with challenges. Training deep neural networks requires substantial computational resources and large amounts of labelled data.
- Overfitting, where the model becomes too specialized to the training data and fails to generalize well to new data, is another common concern. Regularization techniques, such as dropout and weight decay, are often employed to mitigate overfitting.
- Deep learning has significantly advanced the state-of-the-art in various fields, including computer vision, natural language processing, speech recognition, and even areas like drug discovery and genomics. The availability of frameworks and libraries such as **TensorFlow, PyTorch, and Keras** has made it more accessible for researchers and practitioners to apply deep learning techniques in their work.

Pipelines

- A deep learning pipeline refers to the sequential flow of steps involved in building, training, evaluating, and deploying deep learning models. It encompasses the entire process, from:

    1. Data acquisition
    2. Pre-processing
    3. Model training,
    4. Evaluation, and
    5. Deployment.



Pipelines

- Data Acquisition: The pipeline begins with acquiring the required data for training the deep learning model. This can involve collecting data from various sources, such as databases, APIs, or data scraping techniques. The data may be in the form of images, text, audio, or any other relevant format.
- Data Pre-processing: Once the data is acquired, it needs to be pre-processed to ensure its quality and compatibility with the deep learning model. Data pre-processing involves tasks such as cleaning the data, handling missing values, transforming data into a suitable format, and normalizing or scaling the data. For example, in image data, pre-processing may involve resizing, cropping, or normalizing pixel values.
- Model Architecture Design: The next step is to design the architecture of the deep learning model. This includes selecting the appropriate neural network architecture based on the task at hand, such as convolutional neural networks (CNNs) for image-related tasks or recurrent neural networks (RNNs) for sequential data analysis. The architecture design also involves determining the number of layers, the type of activation functions, and other hyperparameters.
- Model Training: With the architecture defined, the deep learning model is trained using the pre-processed data. During training, the model learns to optimize its parameters (weights and biases) to minimize the difference between predicted outputs and ground truth labels. This typically involves iterative forward and backward passes (forward propagation and backpropagation) to update the model's parameters based on calculated gradients.

Pipelines

- Model Evaluation: After training, the model's performance is evaluated using a separate set of data, often referred to as a validation or test set. Evaluation metrics, such as accuracy, precision, recall, or mean squared error, are used to assess the model's performance. The evaluation helps understand how well the model generalizes to unseen data and guides decisions on further iterations or modifications.

- Hyperparameter Tuning: Deep learning models often have various hyperparameters that need to be tuned for optimal performance. Hyperparameters include learning rate, batch size, regularization techniques, optimizer selection, and more. Hyperparameter tuning involves experimenting with different values and configurations to find the best set of hyperparameters that maximize the model's performance.

- Deployment and Inference: Once the model is trained and evaluated, it can be deployed to make predictions on new, unseen data. This involves integrating the model into a production environment, such as a web application or an embedded system, where it can take input data and generate predictions or classifications.

- Model Monitoring and Maintenance: After deployment, it's important to monitor the model's performance and ensure it continues to perform well over time. This may involve monitoring data drift, retraining the model periodically with new data, and updating the model architecture or hyperparameters as needed.

ABO

Pipelines

- Data Quantity and Quality: Deep learning models generally require large amounts of labeled data to achieve high performance. The availability and quality of data play a significant role in the success of deep learning models. It is essential to have diverse and representative data that adequately captures the variations and patterns present in the real-world problem domain.

- Computation and Resources: Deep learning models are computationally intensive and often require powerful hardware resources, such as GPUs (Graphics Processing Units) or TPUs (Tensor Processing Units), to train efficiently. Training deep models can be time-consuming, and the availability of sufficient computational resources can greatly impact the training speed and feasibility.

- Transfer Learning: Transfer learning is a technique commonly used in deep learning, where pre-trained models trained on large-scale datasets are utilized as a starting point for a new, related task. By leveraging the learned representations from pre-training, transfer learning enables faster training and better performance, especially when the target task has limited data.

- Interpretability and Explainability: Deep learning models are often considered as "black boxes" due to their complex nature and large number of parameters. Interpreting the internal workings of deep models and understanding why they make specific predictions or decisions can be challenging. Research in the field of explainable AI aims to develop techniques and tools to provide insights into the decision-making process of deep learning models.

Pipelines

- Ethical Considerations: Deep learning models can inadvertently learn biases present in the training data, leading to biased or unfair decisions. It is important to be aware of the ethical considerations surrounding deep learning, including issues related to data privacy, fairness, and transparency. Efforts are being made to develop frameworks and guidelines for responsible AI development and deployment.

- Continuous Learning: Deep learning models can be updated and refined over time as new data becomes available. Continuous learning, also known as online learning or incremental learning, enables models to adapt and improve with new observations without discarding previous knowledge. This is particularly beneficial in scenarios where data distribution or patterns change over time.

- Hybrid Approaches: Deep learning is often combined with other machine learning techniques to tackle complex problems. Hybrid approaches, such as combining deep learning with traditional machine learning algorithms or rule-based systems, can leverage the strengths of different methodologies to achieve better results.

- Ongoing Research and Advancements: Deep learning is a rapidly evolving field with ongoing research and advancements. New architectures, regularization techniques, optimization algorithms, and model interpretability methods are continually being developed. Staying up to date with the latest research and following community discussions can help practitioners make informed decisions and leverage the latest advancements in deep learning.