MACHINE LEARNING-23CSM241T

UNIT-1

Evolution of Machine Learning:

Machine learning (ML) is a powerful branch of artificial intelligence (AI) that allows computers to learn without explicit programming. Instead of following a set of rigid instructions, ML algorithms can analyze data, identify patterns, and make predictions

Teaching a computer to recognize your friends in photos without showing it every single picture is like training it to learn from examples. Imagine you have a bunch of photos where your friends are tagged with their names. The computer looks at these photos and tries to find patterns. For example, it might notice that one friend has blue eyes and curly hair, while another friend has brown eyes and wears glasses.

Using these patterns, the computer starts to guess who's who in new photos. If it sees someone with blue eyes and curly hair, it might say, "Hey, that looks like Arushi!" But the computer doesn't always get it right at first. It needs lots of practice and feedback to improve, just like how we learn from our mistakes.

This is where machine learning comes in. It's like giving the computer a super smart brain that learns from experience. As the computer sees more and more photos and gets feedback on its guesses, it gets better

at recognizing your friends. It's like **"teaching a robot"** to be a detective, but instead of clues, it uses pictures!

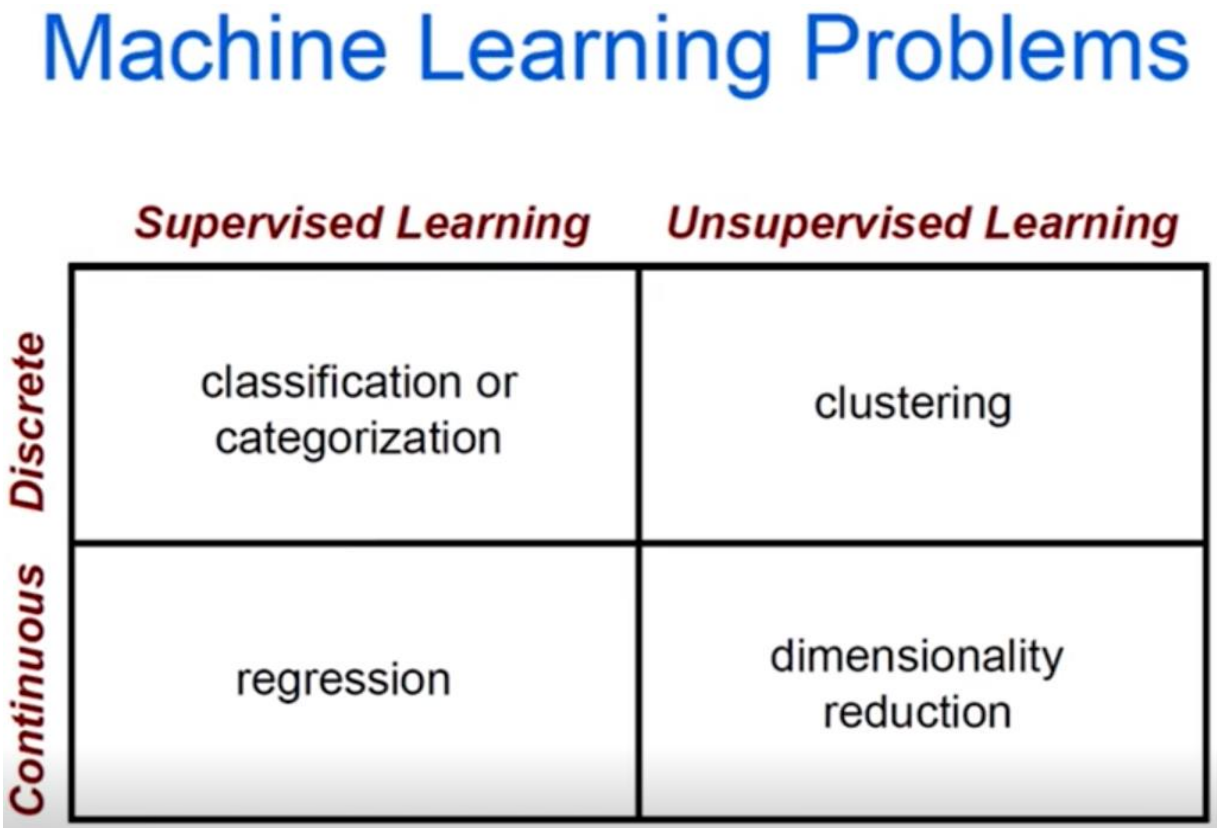
In the big world of artificial intelligence (AI), machine learning is like the secret sauce that helps AI systems become smarter over time. It's what makes AI capable of doing amazing things like understanding speech, playing games, and yes, even recognizing your friends in photos. It's pretty cool how technology can learn and improve, just like we do!

The evolution of machine learning is a captivating story. Early ideas emerged in the 1943s, but limitations in computing power hampered significant progress. With the recent explosion of data and computing muscle, machine learning has taken center stage. Today, it's used everywhere from spam filters in your email to recommending movies you might enjoy. Machine learning is constantly evolving, shaping the future of technology and impacting our lives in profound ways.

Paradigms for ml:

Machine learning (ML) is a dynamic field dedicated to developing methods that enable machines to learn from extensive datasets to enable machines to learn and make predictions. The learning paradigms in ML are categorized based on their resemblance to human interventions, each serving specific purposes and applications. This

dynamic field encompasses various learning paradigms, each with its unique approach to handling data.

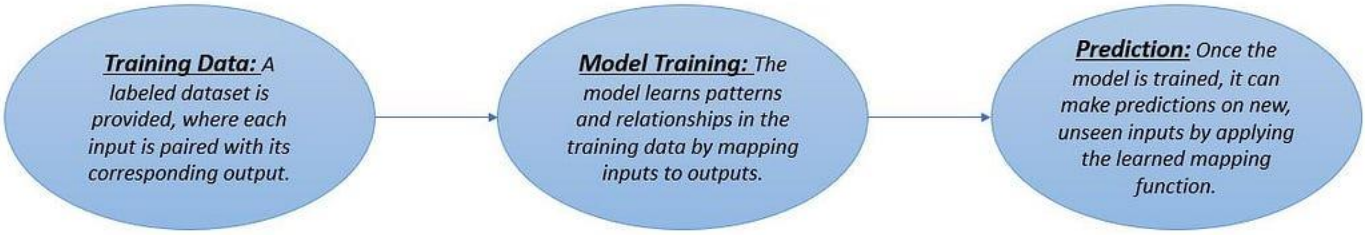


Supervised and Unsupervised learning

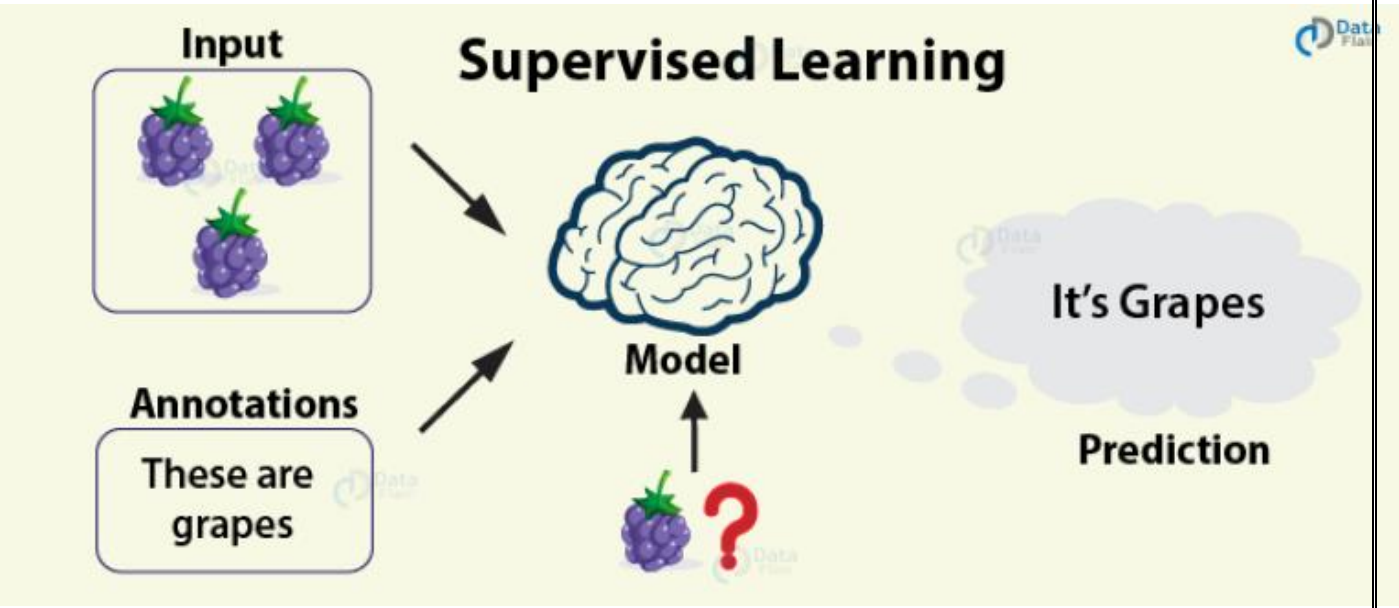
Supervised Learning (SL)

Supervised learning involves labelled datasets, where each data observation is paired with a corresponding class label. Algorithms in supervised learning aim to build a mathematical function that maps

input features to desired output values based on these labeled examples. Common applications include classification and regression.



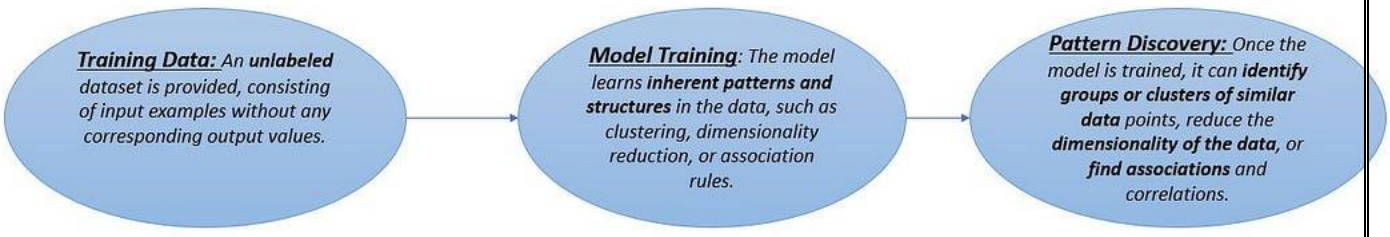
Stages in Supervised Learning



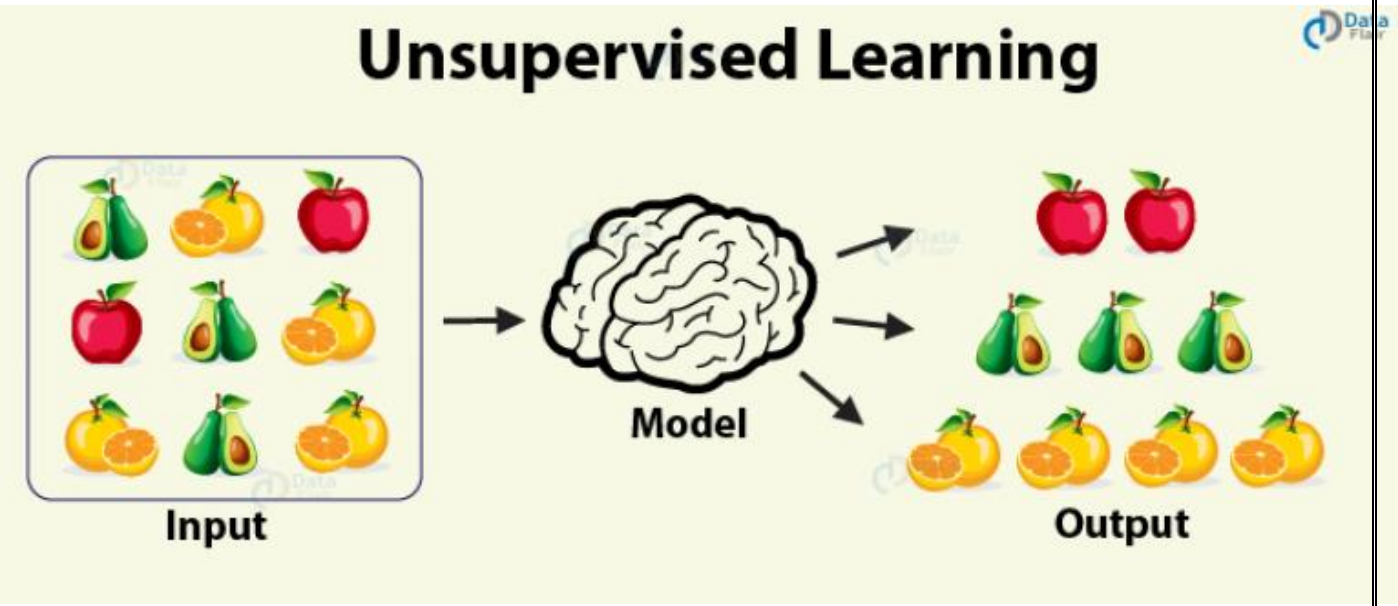
Understanding Supervised Learning pictorially

Unsupervised Learning

In unsupervised learning, algorithms work with unlabeled data to identify patterns and relationships. These methods uncover commonalities within the data without predefined categories. Techniques such as clustering and association rules fall under unsupervised learning.



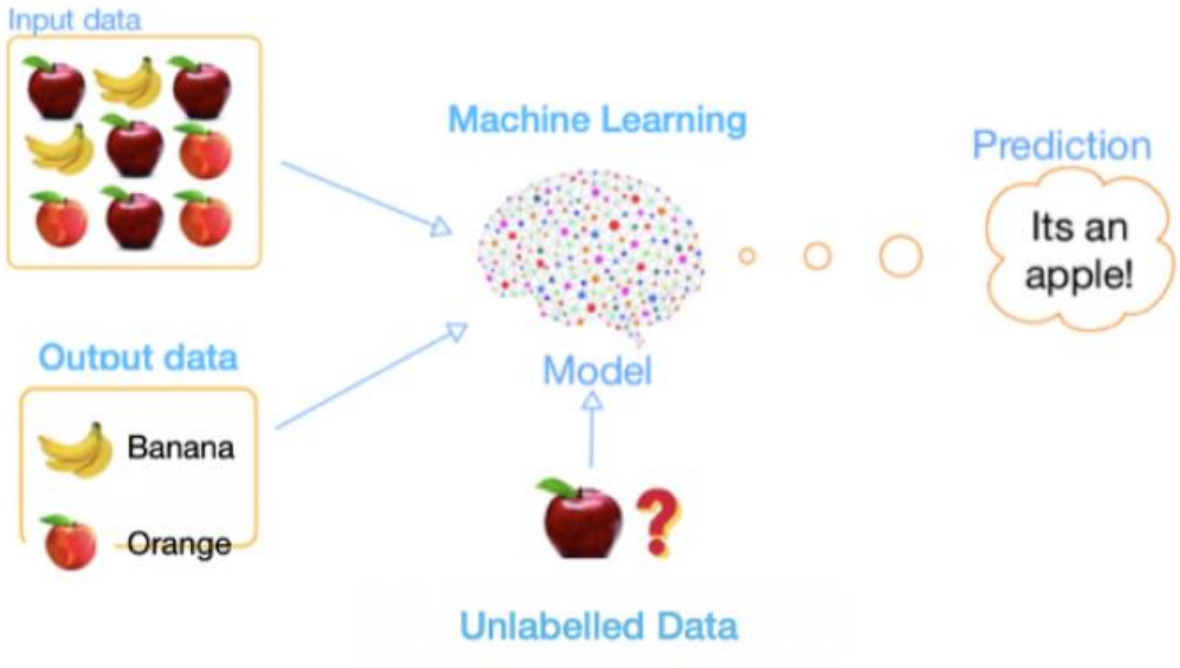
Stages in Unsupervised Learning



Understanding Unsupervised Learning pictorially

Semi-supervised Learning

Semi-supervised learning strikes a balance by combining a small amount of labelled data with a larger pool of unlabeled data. This approach leverages the benefits of both supervised and unsupervised learning paradigms, making it a cost-effective and efficient method for training models when the labeled data is limited.



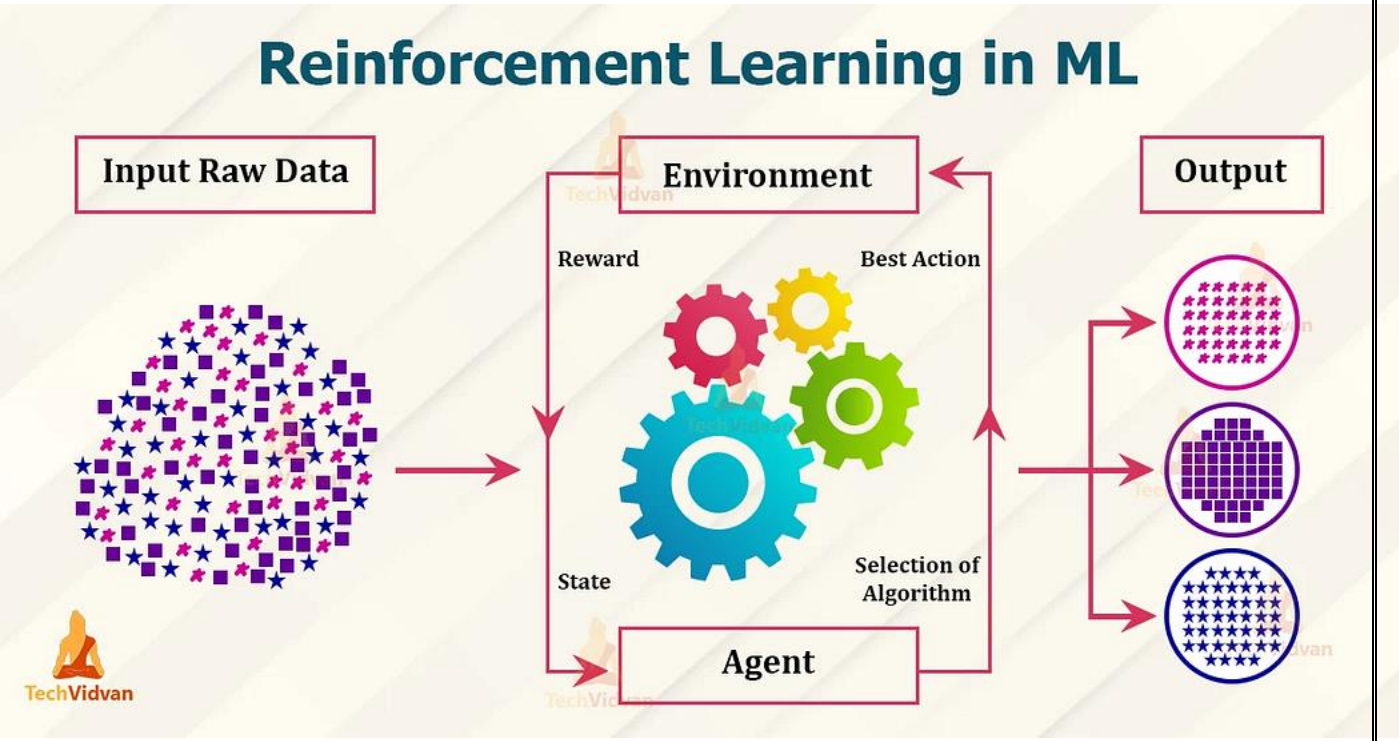
Understanding Semi-supervised Learning pictorially

Self-supervised Learning (SSL)

In scenarios where obtaining high-quality labeled data is challenging, self-supervised learning emerges as a solution. In this paradigm, models are pre-trained using unlabeled data, and data labels are generated automatically during subsequent iterations. SSL transforms unsupervised ML problems into supervised ones, enhancing learning efficiency. This paradigm is particularly relevant with the rise of *large language models*.

Reinforcement Learning

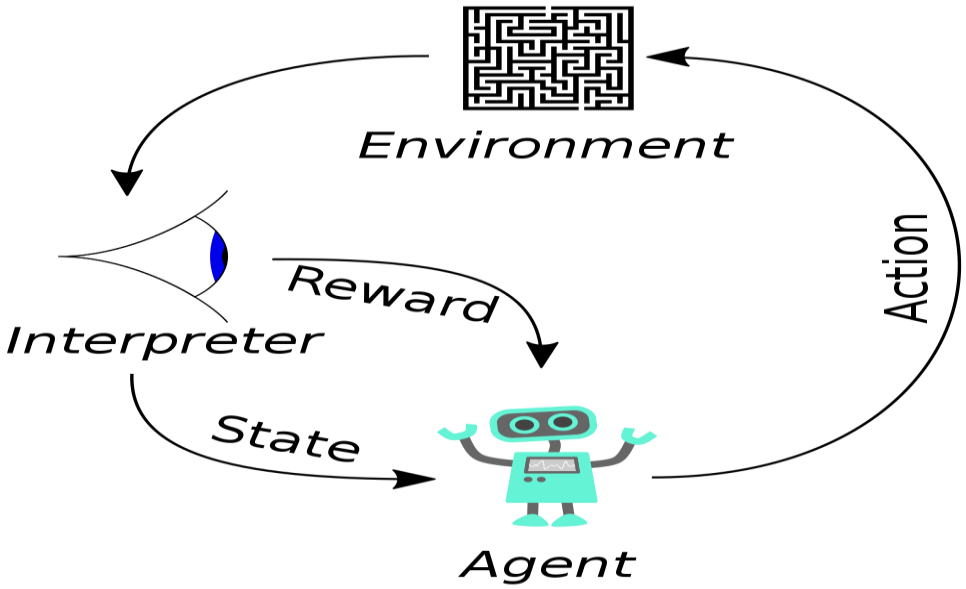
Reinforcement learning focuses on enabling intelligent agents to learn tasks through trial-and-error interactions with dynamic environments. Without the need for labelled datasets, agents make decisions to maximize a reward function. This autonomous exploration and learning approach is crucial for tasks where explicit programming is challenging.



Action-Reward feedback loop: an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent.

Action-Reward Feedback Loop:

Reinforcement learning operates on an action-reward feedback loop, where agents take actions, receive rewards, and interpret the environment's state. This iterative process allows the agent to autonomously learn optimal actions to maximize positive feedback.



Understanding these ML paradigms provides valuable insights into the diverse approaches used to address different types of problems. Each paradigm comes with its strengths and applications, contributing to the versatility of machine learning in various domains.

Learning by Rote:

The meaning of "rote" it self means learning by repetition. The process of repeating something over and over engages the short-term memory and allows us to quickly remember basic things like facts, dates,

names, multiplication tables ,etc. It differs from other forms of learning in that it doesn't require the learner to carefully think about something, and is rather dependent on the act of repetition itself

Even though complete and holistic learning is not dependent on rote learning techniques alone, they do allow students to quickly recall basic facts and laws and master foundational knowledge of a topic in students. Some examples of rote learning in schools can be found in the following:

- Repeating words to instil them in your vocabulary.
- Learning scales in music.
- Memorizing the periodic table.
- Learning the basic laws and formulae in physics and sundry sciences.

Having to memorize the basic facts and principles of a field is an import prerequisite to later analyse and study them. This is where rote learning techniques come in handy and allow you to remember the building blocks of concepts without having to dive deep into them

Rote learning techniques:

Rote learning techniques are aplenty, and they all require time and effort in repetition. The more you repeat for longer periods, the easier

it will be to recall. Even if you only have a few hours to memorize something, the following rote learning techniques will help you remember quickly:

Read it aloud-Read the text out loud with understanding. You can even try it before a mirror, ask a friend to listen to you, or read it out just under your breath. You can experiment with how slow or fast you want to read, how expressive you want to be, and internalize the rhythm of the text. Auditory learners will greatly benefit from this rote learning technique.

Write it down- Writing down the text information after reading is one of the best rote learning techniques. Doing so will help identity difficult passengers and areas that need more practice. If you're preparing for a written exam, this kinesthetic rote technique will serve as a rehearsal and commit the information for easy retrieval

Visualize - Humans are visual creatures and our brains are wired to remember things better with images. For every line and connected phrase, come up with ways to visualize it and remember it. The memory palace can be a useful trick for such rote learning techniques.

Free association-Free association is one of the more intresting rote learning techniques, and a very useful way of remembering things quickly, especially if they are too messy for the traditional rote learning

techniques. The main idea of this method is to combine new information with what you already know in a fun and personal way. For instance, if you're learning the "Circle of Fifths" in music ,you can associate each node to the numbers on the clock, one for each of the 12 notes in music. you are free to form your own associations as you see fit, as long as it helps you to recall the information

Advantages of Rote Learning Techniques:

Rote learning is considered as useful for a variety of reasons. Here are a few:

- Rote learning requires very little analysis.
- With rote, one can remember just about anything over little analysis
- Rote learning allows one to recall information wholly, and even to retain it for life.
- Rote learning makes it easier for people to score who find it difficult to understand or master reading and maths concepts.
- Rote learning can help improve short-term memory.

Disadvantages of rote learning techniques:

On the other hand, there are a few drawbacks of rote learning that you need to be aware of as well

- The repetitive nature of rote learning can become dull.
- One can easily lose focus while rote learning.
- Rote learning is not holistic.
- There is no connection between new and old information with rote learning.
- Rote learning doesn't lead to a deeper understanding of the information.

Learning by Induction:

What is Inductive Learning Algorithm?

Inductive Learning Algorithm (ILA) is an iterative and inductive machine learning algorithm that is used for generating a set of classification rules, which produces rules of the form "IF-THEN", for a set of examples, producing rules at each iteration and appending to the set of rules.

There are basically two methods for knowledge extraction firstly from domain experts and then with machine learning. For a very large amount of data, the domain experts are not very useful and reliable. So we move towards the machine learning approach for this work. To

use machine learning One method is to replicate the expert's logic in the form of algorithms but this work is very tedious, time taking, and expensive. So we move towards the inductive algorithms which generate the strategy for performing a task and need not instruct separately at each step.

Why you should use Inductive Learning?

The ILA is a new algorithm that was needed even when other reinforcement learnings like ID3 and AQ were available.

- The need was due to the pitfalls which were present in the previous algorithms, one of the major pitfalls was the lack of generalization of rules.
- The ID3 and AQ used the decision tree production method which was too specific which were difficult to analyze and very slow to perform for basic short classification problems.
- The decision tree-based algorithm was unable to work for a new problem if some attributes are missing.
- The ILA uses the method of production of a general set of rules instead of decision trees, which overcomes the above problems

Basic Requirements to Apply Inductive Learning Algorithm

1. List the examples in the form of a table 'T' where each row corresponds to an example and each column contains an attribute value.

2. Create a set of m training examples, each example composed of k attributes and a class attribute with n possible decisions.
3. Create a rule set, R, having the initial value false.
4. Initially, all rows in the table are unmarked.

Necessary Steps for Implementation

- **Step 1:** divide the table 'T' containing m examples into n sub-tables (t1, t2,.....tn). One table for each possible value of the class attribute. (repeat steps 2-8 for each sub-table)
- **Step 2:** Initialize the attribute combination count ' j ' = 1.
- **Step 3:** For the sub-table on which work is going on, divide the attribute list into distinct combinations, each combination with 'j ' distinct attributes.
- **Step 4:** For each combination of attributes, count the number of occurrences of attribute values that appear under the same combination of attributes in unmarked rows of the sub-table under consideration, and at the same time, not appears under the same combination of attributes of other sub-tables. Call the first combination with the maximum number of occurrences the max-combination ' MAX'.
- **Step 5:** If 'MAX' == null, increase ' j ' by 1 and go to Step 3.
- **Step 6:** Mark all rows of the sub-table where working, in which the values of 'MAX' appear, as classified.
- **Step 7:** Add a rule (IF attribute = "XYZ" –> THEN decision is YES/ NO) to R whose left-hand side will have attribute names of

the ‘MAX’ with their values separated by AND, and its right-hand side contains the decision attribute value associated with the sub-table.

- **Step 8:** If all rows are marked as classified, then move on to process another sub-table and go to Step 2. Else, go to Step 4. If no sub-tables are available, exit with the set of rules obtained till then.

An example showing the use of ILA suppose an example set having attributes Place type, weather, location, decision, and seven examples, our task is to generate a set of rules that under what condition is the decision.

Example no.	Place type	weather	location	decision
1.	hilly	winter	kullu	Yes
2.	mountain	windy	Mumbai	No
3.	mountain	windy	Shimla	Yes
4.	beach	windy	Mumbai	No
5.	beach	warm	goa	Yes

Example no.	Place type	weather	location	decision
6.	beach	windy	goa	No
7.	beach	warm	Shimla	Yes

Subset – 1

s.no	place type	weather	location	decision
1.	hilly	winter	kullu	Yes
2.	mountain	windy	Shimla	Yes
3.	beach	warm	goa	Yes
4.	beach	warm	Shimla	Yes

Subset – 2

s.no	place type	weather	location	decision
5.	mountain	windy	Mumbai	No

s.no	place type	weather	location	decision
6.	beach	windy	Mumbai	No
7.	beach	windy	goa	No

- **At iteration 1** rows 3 & 4 column weather is selected and rows 3 & 4 are marked. the rule is added to R IF the weather is warm then a decision is yes.
- **At iteration 2** row 1 column place type is selected and row 1 is marked. the rule is added to R IF the place type is hilly then the decision is yes.
- **At iteration 3** row 2 column location is selected and row 2 is marked. the rule is added to R IF the location is Shimla then the decision is yes.
- **At iteration 4** row 5&6 column location is selected and row 5&6 are marked. the rule is added to R IF the location is Mumbai then a decision is no.
- **At iteration 5** row 7 column place type & the weather is selected and row 7 is marked. the rule is added to R IF the place type is beach AND the weather is windy then the decision is no.

Finally, we get the rule set:- Rule Set

- **Rule 1:** IF the weather is warm THEN the decision is yes.
- **Rule 2:** IF the place type is hilly THEN the decision is yes.
- **Rule 3:** IF the location is Shimla THEN the decision is yes.

- **Rule 4:** IF the location is Mumbai THEN the decision is no.
- **Rule 5:** IF the place type is beach AND the weather is windy THEN the decision is no.

Reinforcement Learning:

• It tries to improve its performance of doing the task.

• When a sub-task is accomplished successfully, a reward is given.

• When a sub-task is not executed correctly, obviously no reward is given.

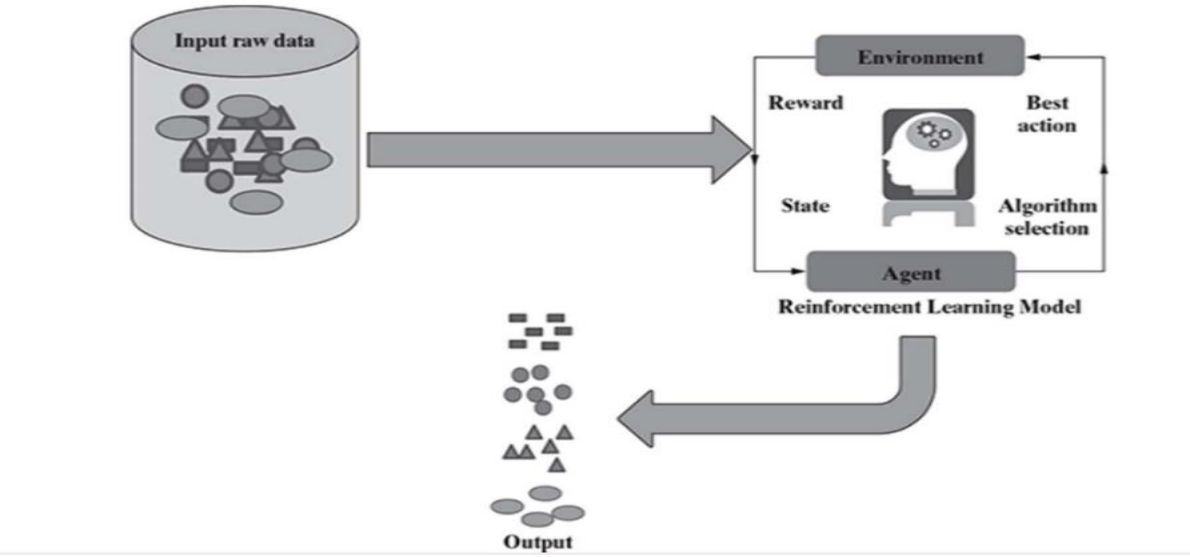
• This continues till the machine is able to complete execution of the whole task.

• This process of learning is known as reinforcement learning.

• One contemporary example of reinforcement learning is self-driving cars.

• The critical information which it needs to take care of are speed and speed limit in different road segments, traffic conditions, road conditions, weather conditions, etc.

• The tasks that have to be taken care of are start/stop, accelerate/decelerate, turn to left / right, etc.



- This type of learning is used when there is no idea about the class or label of a particular data. The model has to do the classification it will get rewarded if the classification is correct, else get punished.
- The model learns and updates itself through reward/punishment
- Model is evaluated by means of the reward function ater it had some time to learn.
- Most complex to understand and apply
- Standard algorithms are
    - Q-Learning
    - Sarsa
- Practical applications are
    - Self-driving cars
    - Intelligent robots
    - AlphaGo Zero[The latest version of DeepMind's AI system playing Go]

Types of Data Matching:

Software tools have been developed to automate the process of data matching.

Large enterprises have a lot of data. And with all that data comes the challenge of keeping it organized and accurate – or more specifically, making sure that their data is being leveraged to its full potential.

One key way to do this is through record or data matching, which is the process of connecting data records that correspond to the same canonical (master) entity.

Most enterprise databases, due to their breadth and depth of data, will have some degree of duplicates or inaccuracies (e.g. in a database of locations, "San Francisco" may be written as "SF", "San Fran", or "SFO"). Data matching tools help to standardize data and improve its quality by identifying these duplicates and linking them to a single, accurate record.

Naturally, software tools have been developed to automate this process. Below, we'll take a look at the various types of data matching tools available, how they work, and some use cases for each.

Types of Data Matching Tools

There are two main types of data matching tools:

1) probabilistic and
2) deterministic.

Probabilistic Data Linkage Tools:

Probabilistic data linkage tools use statistical methods to determine the likelihood that two records refer to the same entity. They work by comparing different fields in the records and assigning a similarity score for each field; the overall similarity score for the two records is then used to make a probabilistic determination of whether they should be linked.

Today, most probabilistic tools employ machine learning algorithms to provide even more accurate results. Regression and natural language processing techniques are often used to automatically identify and extract important features from records, which are then used in the similarity scoring process.

Advantages

- Probabilistic record matching tools can be used on data of any type, **including unstructured data**.
- Real world data tends to be unstructured, and often poorly maintained due to manual data entry.
- They are able to handle many spelling/coding variants and exceptions that would not be easy to cover with a predefined rigid rule set or even a robust dictionary.
- They are generally more accurate than their counterpart, deterministic data linkage tools (more on this below).

Disadvantages:

- They can be more difficult to configure and tune, since there are more parameters that need to be set.
- They require a good amount of data in order to train the machine learning models used for feature extraction and similarity scoring.
- Some of their results are difficult to interpret; since they are based on probabilistic methods, much of the process occurs behind-the-scenes.

Deterministic Data Matching Tools:

Deterministic data matching tools, on the other hand, use rule-based methods to connect records. That is, they compare different fields in the records using a system like RegEx and look for exact matches; if two fields match, then the records are linked.

Since deterministic tools use a predetermined set of rules, they are generally much easier to configure than their probabilistic counterparts. However, this also means that they can be less accurate, since they may miss some relationships that don't fit the rules.

Deterministic data matching tools are often used in cases where data is highly structured and well-defined; for example, when linking records from two different databases that use the same schema. By contrast, probabilistic data linkage tools are better suited for data that is unstructured or has many different schemas.

Advantages

- They are much easier to configure and tune than probabilistic data linkage tools.
- They can be used on data of any type, including unstructured data.
- You don't need much data in order to use them, since they don't require training data for machine learning models.

Disadvantages

- They can be less accurate than probabilistic data matching tools, since they may miss some relationships that don't fit the rules.
- They are often less flexible than probabilistic data matching tools, since they can only compare fields in a predetermined way.
- Some of their results are difficult to interpret; since they are based on deterministic methods, much of the process occurs behind-the-scenes.

Uses of Data Matching Tools

Record matching tools can be used for a variety of tasks, including:

- **Data deduplication** – identifying and removing duplicate records from your database as a specific form of data quality assurance.
- **Data enrichment** – combining your data with other data sources in order to enrich it.

- **Data integration** – connecting different data sources that use different schemas.
- **Fraud detection** – identifying records that are likely to be fraudulent, based on their similarity to other records in your database or specific features of known examples.
- **Building customer 360 profiles:** connecting customer data from different sources (e.g. social media, website interactions, customer service interactions) in order to get a complete view of the customer.

In addition to direct benefits to revenue, there are also longer-term benefits to be gained from using record matching tools.

For example, by linking customer data across different channels, you can develop a deeper understanding of customer behavior. This, in turn, can lead to better targeted marketing campaigns and improved customer retention rates. Perhaps the company is gearing up to sell to a large acquirer. In this case, having accurate and up-to-date data is critical in order to get the best price for the business.

Machine Learning in Data Matching Tools:

Probabilitic reocrd linkage tools make use of machine learning trained for specific tasks. ML algorithms are what handles confidence scoring of records. In this case we are dealing with machine learning rather than the current wave of AI tools

Machine learning employed for data matching does its task very well, making it a very reliable way of reducing manual work and improving the quality of your data.

Stages in Machine Learning:

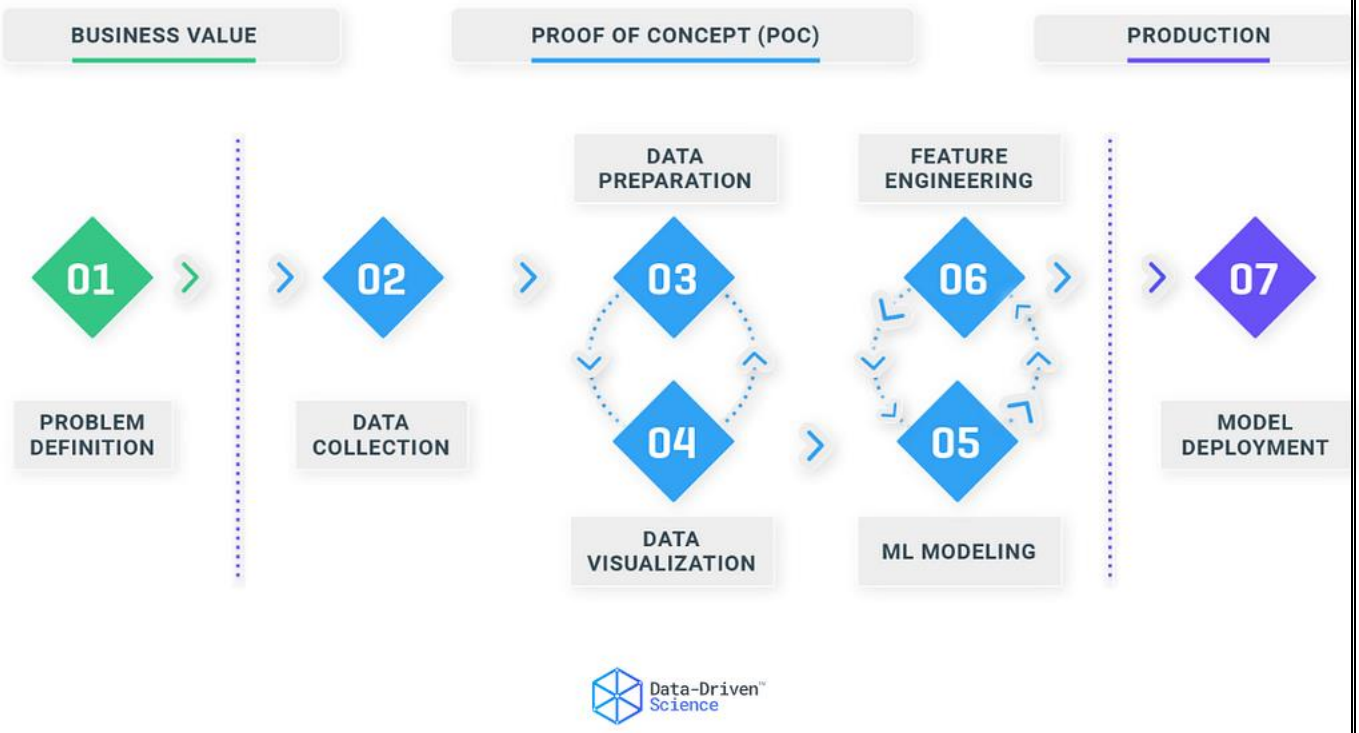
The goal of the Seven Stages framework is to break down all necessary tasks in Machine Learning and organize them in a logical way. At the end, the framework acts as a general process that can be universally applied to any project independently of industry and type of business.

Data-Driven Science (DDS) will also use that framework for its upcoming comprehensive Machine Learning online course published on Udemy — stay tuned. We will go deep into each stage and give you everything that is needed to complete Data Science projects successfully.

The 7 Stages of Machine Learning are:

1. Problem Definition
2. Data Collection
3. Data Preparation
4. Data Visualization
5. ML Modeling

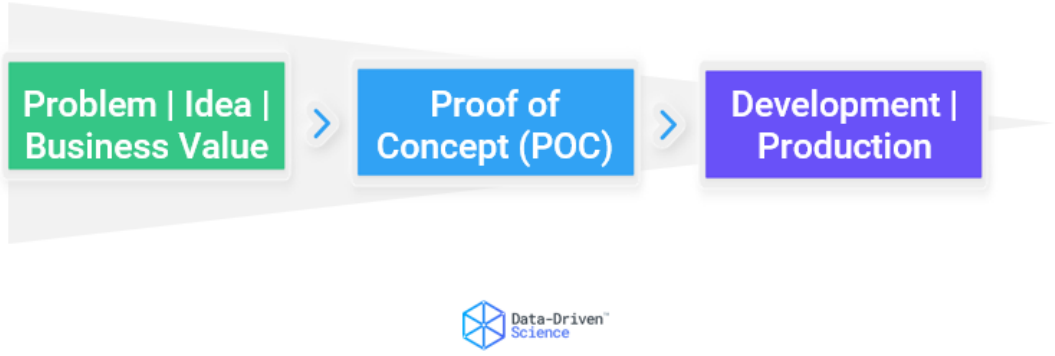
6. Feature Engineering
7. Model Deployment



These 7 stages are the key steps in our framework. We have categorized them additionally into groups to get a better understanding of the larger picture.

The stages are grouped into 3 phases:

1. Business Value
2. Proof of Concept (POC)
3. Production



Phase 1 — Business Value

It is absolutely crucial to adopt a business mindset when thinking sabout a problem that should be solved with Machine Learning — defining customer benefits and creating business impact is top priority. Domain expertise and knowledge is also essential as the true power of data can only be harnessed if the domain is well known and understood.

Phase 2 — Proof of Concept (POC)

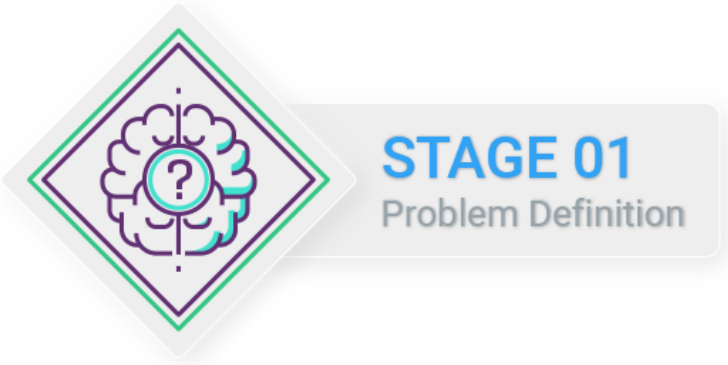
Proof of Concept (POC) is the most comprehensive part of our framework. From Data Collection to Feature Engineering, 5 stages of our ML framework are included here. Core of any POC to test an idea in terms of its feasibility and value to the business. Also, questions around performance and evaluation metrics are answered in that phase. Only a strong POC that delivers business value and is feasible allows one putting the ML Model into production.

Phase 3 — Production

In the third phase, one is taking the ML model and scaling it. The goal is to integrate Machine Learning into a business process solving a problem with a superior solution compared to, for example, traditional programming. The process of taking a trained ML model and making its predictions available to users or other systems is known as model deployment. Lastly, it is also essential to iterate on the ML model over time to improve it.

7 Stages of Machine Learning

1. Problem Definition



The first stage in the DDS Machine Learning Framework is to define and understand the problem that someone is going to solve. Start by analyzing the goals and the *why* behind a particular problem statement. Understand the power of data and how one can use it to

make a change and drive results. And asking the right questions is always a great start.

Few possible questions:

- What is the business?
- Why does the problem need to be solved?
- Is a traditional solution available to solve the problem?
- If probabilistic in nature, then does available data allow to model it?
- What is a measurable business goal?

2. Data Collection

Once the goal is clearly defined, one has to start getting the data that is needed from various available data sources.

At this stage, some of the questions worth considering are:

- What data do I need for my project?
- Where is that data available?
- How can I obtain it?
- What is the most efficient way to store and access all of it?

There are many different ways to collect data that is used for Machine Learning. For example, focus groups, interviews, surveys, and internal usage & user data. Also, public data can be another source and is usually free. These include research and trade associations such as banks, publicly-traded corporations, and others. If data isn't publicly available, one could also use web scraping to get it (however, there are some legal restrictions).

3. Data Preparation

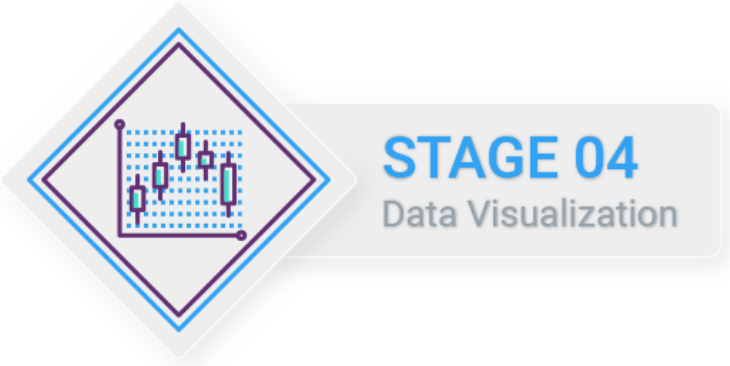
The third stage is the most time-consuming and labor-intensive. Data Preparation can take up to 70% and sometimes even 90% of the overall project time. But what is the purpose of this stage?

Well, the type and quality of data that is used in a Machine Learning model affects the output considerably. In Data Preparation one explores, pre-processes, conditions, and transforms data prior to modeling and analysis. It is absolutely essential to understand the data, learn about it, and become familiar before moving on to the next stage.

Some of the steps involved in this stage are:

- Data Filtering
- Data Validation & Cleansing
- Data Formatting
- Data Aggregation & Reconciliation

4. Data Visualization



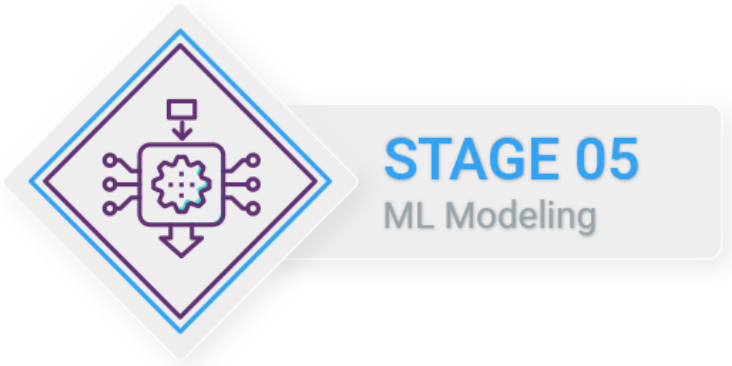
Data Visualization is used to perform Exploratory Data Analysis (EDA). When one is dealing with large volumes of data, building graphs is the best way to explore and communicate findings. Visualization is an incredibly helpful tool to identify patterns and trends in data, which leads to clearer understanding and reveals important insights. Data Visualization also helps for faster decision making through the graphical illustration.

Here are some common ways of visualization:

- Area Chart
- Bar Chart
- Box-and-whisker Plots
- Bubble Cloud
- Dot Distribution Map

- Heat Map
- Histogram
- Network Diagram
- Word Cloud

5. ML Modeling

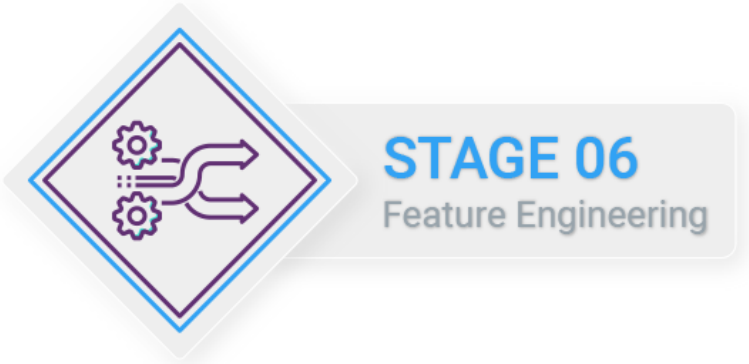


Finally, this is where *'the magic happens'*. Machine Learning is finding patterns in data, and one can perform either supervised or unsupervised learning. ML tasks include regression, classification, forecasting, and clustering.

In this stage of the process one has to apply mathematical, computer science, and business knowledge to train a Machine Learning algorithm that will make predictions based on the provided data. It is a crucial step that will determine the quality and accuracy of future

predictions in new situations. Additionally, ML algorithms help to identify key features with high predictive value.

6. Feature Engineering



Machine Learning algorithms learn recurring patterns from data. Carefully engineered features are a robust representation of those patterns.

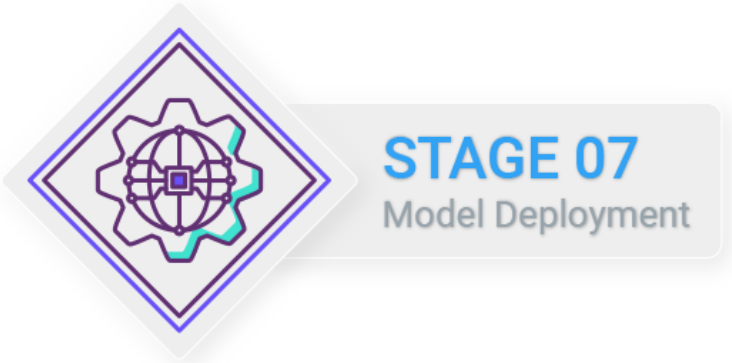
Feature Engineering is a process to achieve a set of features by performing mathematical, statistical, and heuristic procedures. It is a collection of methods for identifying an optimal set of inputs to the Machine Learning algorithm. Feature Engineering is extremely important because well-engineered features make learning possible with simple models.

Following are the characteristics of good features:

- Represents data in an unambiguous way

- Ability to captures linear and non-linear relationships among data points
- Capable of capturing the precise meaning of input data
- Capturing contextual details

7. Model Deployment



The last stage is about putting a Machine Learning model into a production environment to make data-driven decisions in a more automated way. Robustness, compatibility, and scaleability are important factors that should be tested and evaluated before deploying a model. There are various ways such as Platform as a Service (PaaS) or Infrastructure as a Service (IaaS). For containerized applications, one can use container orchestration platform such as Kubernetes to rapidly scale the number of containers as demand shifts.

Another important part of the last stage is iteration and interpretation. It is critical to constantly optimize the model and pressure test the

results. At the end, Machine Learning has to provide value to the business and make a positive impact. Therefore, monitoring the model in production is key.

Conclusion

This was an overview about '**The 7 Stages of Machine Learning**' — a framework that helps to structure the typical process of a ML project. The idea is to equip practitioners with a template that can be universally applied and simplifies the process from idea to implementation.

Data acquisition:

Data acquisition, or DAQ, is the cornerstone of machine learning. It is essential for obtaining high-quality data for model training and optimizing performance. Data-centric techniques are becoming more and more important across a wide range of industries, and DAQ is now a vital tool for improving productivity, preserving quality, and stimulating innovation.

What is Data Acquisition?

"The process of collecting and storing data for machine learning from a variety of sources is known as data acquisition(DAQ)."

The procedure entails gathering, examining, and using crucial data to guarantee precise measurements, instantaneous observation, and

knowledgeable decision-making. Sensors, measuring devices, and a computer work together in DAQ systems to transform physical parameters into electrical signals, condition and amplify those signals, and then store them for analysis.

What is Data Acquisition in Machine Learning?

In machine learning, "data acquisition" refers to the procedure of obtaining and compiling data from diverse sources in order to test and train machine learning models. In order to enable computers and software to manipulate and modify signals from real-world occurrences, this technique entails digitizing such signals. Data Acquisition aims to get a complete and representative dataset that successfully captures the patterns and changes in the data that are crucial for productive machine learning results.

The process of acquiring data also include taking the variable into account that affect its quality and utility, such as volume, velocity, anddiversity.

Successful machine learning begins with data collecting, which supplies the raw information required to train models and make defensible conclusions. The gathering of high-quality data is essential for providing machine learning algorithms with the necessary input to enable them to learn and perform better.

What Does a DAQ System Measure?

A Data Acquisition (DAQ) system is capable of measuring several physical parameters, such as:

- **Temperature:** Temperature can be measured using RTDs, thermistors, or thermocouples in DAQ systems.
- **Pressure:** In a variety of settings, including industrial operations and medical equipment, pressure is measured using pressure sensors.
- **Voltage:** Power systems, electronics, and electrical engineering all depend on the ability of DAQ devices to monitor the voltage levels in electrical circuits.
- **Current:** DAQ systems can measure current flow using current sensors or shunts. Current measurement is essential in electrical systems.
- **Strain and Pressure:** Deformation and pressure in materials are measured using strain gauges and pressure sensors, which is crucial for material science and structural health monitoring.
- **Shock and Vibration:** In a variety of fields, including mechanical, aeronautical, and civil engineering, accelerometers and vibration sensors are used to monitor shock, vibration, and acceleration.
- **RPM, Angle, and Discrete Events**: DAQ systems are crucial for robotics, automation, and mechanical systems because they can measure rotational speed, angle, and discrete events.

- **Distance and Displacement:** Ultrasonic, laser, and encoder sensors are among the sensors that DAQ systems can use to detect distance and displacement.
- **Weight**: Measuring weight is crucial for a number of applications, including quality control, logistics, and industrial automation.

Components of Data Acquisition System

To understand how data is selected and processed, a data acquisition system consists of below key basic components: sensors, measuring instruments, and a computer.

1**. Sensors:** Sensors are devices that quantify and translate physical parameters like voltage, pressure, or temperature into electrical impulses. Later, these signals are sent to the measuring devices for additional analysis.

2.Signal Conditioner: Signal conditioning is the process of improving raw sensor signals so they can be reliably understood. To make sure that the signals are dependable, clear, and compatible with the rest of the system, signal conditioning procedures include isolation, amplification, and filtering.

- **Amplification:** It helps in improving accuracy by maximizing the signal strength
- **Filtering:** Filters extra and unwanted noise from the signal
- **Isolation:** Helps in separating sensor from DAQ system.

3. Analog-to-digital Converter: After the signals are conditioned, they must be translated into a digital format that computers can comprehend using an analog-to-digital converter (ADC). The

continuous analog signals are transformed into discrete digital values so that the system can process and store them.

4. Data Logger: The data logger serves as the operation's central nervous system. A device or software program known as a data logger is responsible for managing incoming data, controlling the acquisition process, and storing it for subsequently analysis.

5. Data Processing Unit: After receiving data from ADC, the system has dedicated card to process the signals like sampling, buffering and Data Transfer.

6. Data Storage : Acquired data is stored in the computer's memory for real-time monitoring.

The physical parameters are measured using sensors, which convert the physical signals into electrical signals. The signals are then conditioned, amplified, and converted into digital data using analog-to-digital converters (ADCs). The digital data is then processed, analyzed, and stored using computers and software.

What are the Major Purposes of Data Acquisition?

Although there are many different and important reasons, some of the most important ones are as follows:

- **Long-term analysis and trend detection:** Long-term analysis are made possible by data acquisition systems, which make it possible to log, capture, and store measurement of data over an extended period of time.
- **Measurement that is accurate and dependable**: DAQ systems and equipment provide measurement that is accurate and

dependable, enabling uses like optical analysis and light intensity monitoring.

- **Industry Leading devices:** DAQ systems and devices are widely used, connecting to a variety of sensors and collaborating with contemporary computers, which makes them an excellent option for scientists and researchers looking for accurate data.
- **Enhanced productivity and dependability of machines:** Data capture gives an organization more control over its operations and enables quicker reaction to potential breakdowns, maximizing procedure optimization.
- **Faster problem analysis and resolution**: Real-time data acquisition systems allow measurements to be produced and shown instantly, which allows personnel to respond to issues more quickly and get the machine operating at peak efficiency in less time.
- **Reduction of data redundancy**: DAQ systems let businesses operate without interference from extraneous data by making it easier to analyze the information they have collected.

What are the Different Data Acquisition Options?

Devices like sensors, transducers, and other devices can provide data, which data acquisition (DAQ) systems are made to measure, record, and analyze. Selecting the right DAQ system relies on the requirements and particular application. There are various types of DAQ systems, each with advantages and disadvantages of their own. The following are a few of the several options for acquiring data:

- **Data loggers:** These are compact, lightweight gadgets with extended data recording capabilities. They are frequently employed in applications like industrial automation and environmental monitoring where data collection in the field is required.
- **Data acquisition devices: These** are plug-and-play items that can be linked via USB or other interfaces to a computer. They are perfect for projects where requirements don't alter because they offer set functionality.
- **Data acquisition systems:** These are modular systems that can be set up to accommodate certain measurement requirements. They are perfect for complex systems that need several channels and high-speed data gathering because of their tremendous versatility.
- **Computer-Connected DAQ Modules:** These DAQ systems provide an affordable way to get data by connecting to a computer. Comparing them to stand-alone systems, they are frequently lighter and smaller.
- **Stand-Alone or Portable DAQ Systems:** These are DAQ systems that record and analyze data without the need for extra hardware because they come with an integrated computer. They are frequently employed in situations when using a computer is either inconvenient or not possible.
- **Modular DAQ Systems:** These systems are composed of a chassis and several modules that are movable and addable. They

are very flexible and perfect for applications that need to acquire data quickly over several channels.

- **PXIe Modular DAQ Systems**: These are high-performance DAQ systems that link several modules together via the PXIe (PCI Express) interface. They are perfect for applications that demand low latency and high channel counts because they provide fast data capture.

Types of Data Acquisition Sources

- **Sensors**: Convert physical parameters to electrical signals.
- **IoT devices**: Collect data from remote sources using secure communication channels and encryption.
- **Network devices**: Collect data from network devices using secure communication channels and encryption.
- **Manual data entry**: Implement robust access control mechanisms, authentication, and authorization processes to increase the security of manual data entry.
- **Experiments**: Collect primary data through experiments, such as wet lab experiments like gene sequencing.
- **Observations**: Collect primary data through observations, such as surveys, sensors, or in situ collection.
- **Simulations**: Collect primary data through simulations, such as theoretical models like climate models.
- **Scraping or compiling**: Collect primary data through web scraping, text mining, or compiling data from various sources.

- **Institutionalized data banks**: Collect secondary data from institutionalized data banks, such as census or gene sequences.
- **Published datasets**: Collect secondary data from published datasets, such as those found on Kaggle, GitHub, or UCI Machine Learning Repository.
- **APIs**: Collect secondary data through application programming interfaces (APIs), which allow clients to request data from a website's server.
- **Surveys**: Collect primary data through surveys, which can be online or offline.

Importance of Data Acquisition in Machine Learning

Data Acquisition (DAQ) is definitely the most fundamental task that precedes any machine learning project and should not be overlooked. Here's why it holds such importance:

- **Fuel for Learning:** In contrast to the biological organisms, which can sense the objects, the machine learning models are basically recognition-of-patterns technology. Information and intelligence of the model would not be valid if data quality is not up to standard and this affects the model's ability to learn as well as make credible predictions. DAQ just thus guarantees that you have the right "fuel" to be the engine of your model's learning.
- **Quality In, Quality Out:** The sentence "**garbage in garbage out"** illustrates this best. Just as if your model inherits data issues such as inaccuracy, incompleteness, or irrelevancy, it will transmit these flaws into your model unfortunately. DAQ that is successful,

supplies you with data whose quality is great and leads to formation of powerful, and reliable machine learning models.

- **Relevance is Key:** DAQ is what makes you gather data of that problem your model want to learn. The higher the relevance of the data, the greater your model will perceive the dependency between the essence and, therefore, will make precise conclusions.
- **Shaping Model Performance:** You end up with the amount of data you collect for your model, which most often affects the model's performance. An important case in machine learning is when the algorithms need massive data sets in order to learn properly. Expert DAQ strategies allow to collect considerable amount of data for you to train your model so that you can just correctly generalize and answer the questions that it hasn't seen.

The Measurement Process

The measurement process is determining how many units of a specific quantity or quality needs to be measured object. It is an essential procedure in many disciplines, such as science, engineering, building, and daily life. There are various steps to the measurement process, which include:

- **Define the quantity that has to be measured:** The defining of the quantity to be measured is the first step in the measurement process, which also always includes a comparison with a known quantity of the same kind. Finding the physical quantity or attribute that has to be measured is part of this process.

- **Comparing the object or quantity:** The object is compared to a known quantity of the same kind.
- **Transduction**: The quantity or item to be measured is "transduced" into an analogous measurement signal if it cannot be directly compared.
- **Transmission and processing of the signal**: To generate a measurement reading, the physical signal is routed through the system and subjected to processing.
- **Calibration:** The process of obtaining the reference signal from items with known quantities is known as calibration.
- **Quantization:** The measurement is quantized by counting or splitting the signal into equal and known-sized pieces, and the physical signal is compared with the reference signal.

Data Acquisition Tools

Tools for gathering, analyzing, and recording data from a variety of sensors, instruments, or devices are software and hardware systems known as data acquisition tools. **Data Acquisition Tools** are useful in scientific research, industrial automation, engineering, and other domains where data gathering and processing are critical. Few Tools for Acquiring Data are:

- **DriveSpy:** A data collection tool for Windows operating systems created by Digital Intelligence Forensic Solutions.
- **DewesoftX**: A software suite for acquiring and analyzing data that provides strong tools for these tasks.

- **LabVIEW:** A popular software program used in many different industries that offers tools for data collection, processing, and visualization.
- **Catman**: A data acquisition software package that offers tools for data acquisition, analysis, and visualization, and is commonly used in industrial automation and engineering.
- **Matlab**: A software package that provides tools for data acquisition, analysis, and visualization, and is widely used in various industries.
- **FlexPro**: A data acquisition software package that offers tools for data acquisition, analysis, and visualization, and is commonly used in industrial automation and engineering.

Conclusion

In conclusion, Data Acquisition (DAQ) is the crucial first step in building successful machine learning models. It involves gathering high-quality, relevant data to train your models and achieve optimal performance. By following the best practices outlined above, you can ensure your DAQ process is efficient and effective, laying a strong foundation for your machine learning project.

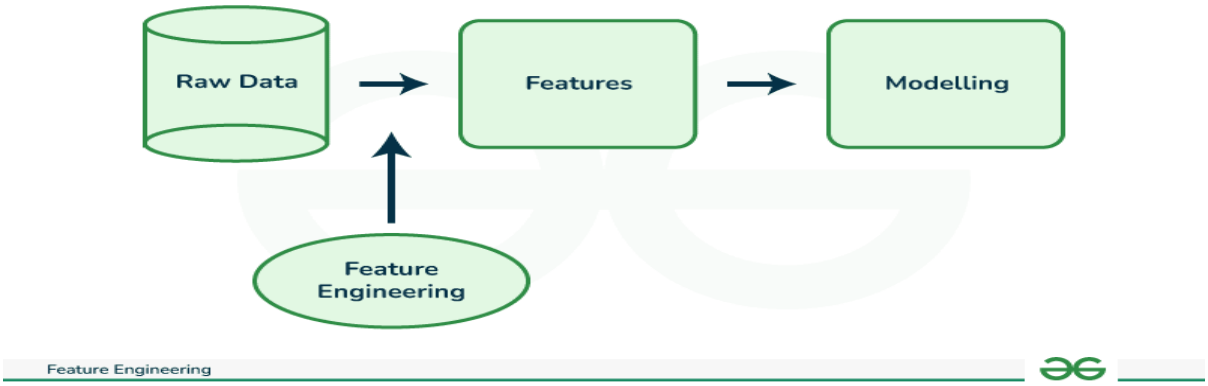
Feature Engineering:

Feature Engineering is the process of creating new features or transforming existing features to improve the performance of a machine-learning model. It involves selecting relevant information from raw data and transforming it into a format that can be easily understood by a model. The goal is to improve model accuracy by providing more meaningful and relevant information.

What is Feature Engineering?

Feature engineering is the process of **transforming raw data into features that are suitable for machine learning models**. In other words, it is the process of selecting, extracting, and transforming the most relevant features from the available data to build more accurate and efficient machine learning models.

The success of machine learning models heavily depends on the quality of the features used to train them. Feature engineering involves a set of techniques that enable us to create new features by combining or transforming the existing ones. These techniques help to highlight the most important patterns and relationships in the data, which in turn helps the machine learning model to learn from the data more effectively.



What is a Feature?

In the context of machine learning, a feature (also known as a variable or attribute) is an individual measurable property or characteristic of a data point that is used as input for a machine learning algorithm. Features can be numerical, categorical, or text-based, and they represent different aspects of the data that are relevant to the problem at hand.

- For example, in a dataset of housing prices, features could include the number of bedrooms, the square footage, the location, and the age of the property. In a dataset of customer demographics, features could include age, gender, income level, and occupation.
- The choice and quality of features are critical in machine learning, as they can greatly impact the accuracy and performance of the model.

Need for Feature Engineering in Machine Learning?

We engineer features for various reasons, and some of the main reasons include:

- **Improve User Experience:** The primary reason we engineer features is to enhance the user experience of a product or service. By adding new features, we can make the product more intuitive, efficient, and user-friendly, which can increase user satisfaction and engagement.
- **Competitive Advantage:** Another reason we engineer features is to gain a competitive advantage in the marketplace. By offering unique and innovative features, we can differentiate our product from competitors and attract more customers.
- **Meet Customer Needs:** We engineer features to meet the evolving needs of customers. By analyzing user feedback, market trends, and customer behavior, we can identify areas where new features could enhance the product's value and meet customer needs.
- **Increase Revenue:** Features can also be engineered to generate more revenue. For example, a new feature that streamlines the checkout process can increase sales, or a feature that provides additional functionality could lead to more upsells or cross-sells.
- **Future-Proofing:** Engineering features can also be done to future-proof a product or service. By anticipating future trends and potential customer needs, we can develop features that ensure the product remains relevant and useful in the long term.

Processes Involved in Feature Engineering

Feature engineering in Machine learning consists of mainly 5 processes: Feature Creation, Feature Transformation, Feature

Extraction, Feature Selection, and Feature Scaling. It is an iterative process that requires experimentation and testing to find the best combination of features for a given problem. The success of a machine learning model largely depends on the quality of the features used in the model.

1. Feature Creation

Feature Creation is the process of generating new features based on domain knowledge or by observing patterns in the data. It is a form of feature engineering that can significantly improve the performance of a machine-learning model.

Types of Feature Creation:

1. **Domain-Specific:** Creating new features based on domain knowledge, such as creating features based on business rules or industry standards.
2. **Data-Driven:** Creating new features by observing patterns in the data, such as calculating aggregations or creating interaction features.
3. **Synthetic:** Generating new features by combining existing features or synthesizing new data points.

Why Feature Creation?

1. **Improves Model Performance:** By providing additional and more relevant information to the model, feature creation can increase the accuracy and precision of the model.

2. **Increases Model Robustness:** By adding additional features, the model can become more robust to outliers and other anomalies.
3. **Improves Model Interpretability:** By creating new features, it can be easier to understand the model's predictions.
4. **Increases Model Flexibility:** By adding new features, the model can be made more flexible to handle different types of data.

2. Feature Transformation

Feature Transformation is the process of transforming the features into a more suitable representation for the machine learning model. This is done to ensure that the model can effectively learn from the data.

Types of Feature Transformation:

1. **Normalization:** Rescaling the features to have a similar range, such as between 0 and 1, to prevent some features from dominating others.
2. **Scaling:** Scaling is a technique used to transform numerical variables to have a similar scale, so that they can be compared more easily. Rescaling the features to have a similar scale, such as having a standard deviation of 1, to make sure the model considers all features equally.
3. **Encoding:** Transforming categorical features into a numerical representation. Examples are one-hot encoding and label encoding.

4. **Transformation:** Transforming the features using mathematical operations to change the distribution or scale of the features. Examples are logarithmic, square root, and reciprocal transformations.

Why Feature Transformation?

1. **Improves Model Performance:** By transforming the features into a more suitable representation, the model can learn more meaningful patterns in the data.
2. **Increases Model Robustness:** Transforming the features can make the model more robust to outliers and other anomalies.
3. **Improves Computational Efficiency:** The transformed features often require fewer computational resources.
4. **Improves Model Interpretability:** By transforming the features, it can be easier to understand the model's predictions.

3. Feature Extraction

Feature Extraction is the process of creating new features from existing ones to provide more relevant information to the machine learning model. This is done by transforming, combining, or aggregating existing features.

Types of Feature Extraction:

1. **Dimensionality Reduction:** Reducing the number of features by transforming the data into a lower-dimensional space while retaining important information. Examples are PCA and t-SNE.

2. **Feature Combination:** Combining two or more existing features to create a new one. For example, the interaction between two features.
3. **Feature Aggregation:** Aggregating features to create a new one. For example, calculating the mean, sum, or count of a set of features.
4. **Feature Transformation:** Transforming existing features into a new representation. For example, log transformation of a feature with a skewed distribution.

Why Feature Extraction?

1. **Improves Model Performance:** By creating new and more relevant features, the model can learn more meaningful patterns in the data.
2. **Reduces Overfitting:** By reducing the dimensionality of the data, the model is less likely to overfit the training data.
3. **Improves Computational Efficiency:** The transformed features often require fewer computational resources.
4. **Improves Model Interpretability:** By creating new features, it can be easier to understand the model's predictions.

4. Feature Selection

Feature Selection is the process of selecting a subset of relevant features from the dataset to be used in a machine-learning model. It is an important step in the feature engineering process as it can have a significant impact on the model's performance.

Types of Feature Selection:

1. **Filter Method:** Based on the statistical measure of the relationship between the feature and the target variable. Features with a high correlation are selected.
2. **Wrapper Method:** Based on the evaluation of the feature subset using a specific machine learning algorithm. The feature subset that results in the best performance is selected.
3. **Embedded Method:** Based on the feature selection as part of the training process of the machine learning algorithm.

Why Feature Selection?

1. **Reduces Overfitting:** By using only the most relevant features, the model can generalize better to new data.
2. **Improves Model Performance:** Selecting the right features can improve the accuracy, precision, and recall of the model.
3. **Decreases Computational Costs:** A smaller number of features requires less computation and storage resources.
4. **Improves Interpretability:** By reducing the number of features, it is easier to understand and interpret the results of the model.

5. Feature Scaling

Feature Scaling is the process of transforming the features so that they have a similar scale. This is important in machine learning because the scale of the features can affect the performance of the model.

Types of Feature Scaling:

1. **Min-Max Scaling:** Rescaling the features to a specific range, such as between 0 and 1, by subtracting the minimum value and dividing by the range.
2. **Standard Scaling:** Rescaling the features to have a mean of 0 and a standard deviation of 1 by subtracting the mean and dividing by the standard deviation.
3. **Robust Scaling:** Rescaling the features to be robust to outliers by dividing them by the interquartile range.

Why Feature Scaling?

1. **Improves Model Performance:** By transforming the features to have a similar scale, the model can learn from all features equally and avoid being dominated by a few large features.
2. **Increases Model Robustness:** By transforming the features to be robust to outliers, the model can become more robust to anomalies.
3. **Improves Computational Efficiency:** Many machine learning algorithms, such as k-nearest neighbors, are sensitive to the scale of the features and perform better with scaled features.
4. **Improves Model Interpretability:** By transforming the features to have a similar scale, it can be easier to understand the model's predictions.

What are the Steps in Feature Engineering?

The steps for feature engineering vary per different Ml engineers and data scientists. Some of the common steps that are involved in most machine-learning algorithms are:

1. **Data Cleansing**
    - Data cleansing (also known as data cleaning or data scrubbing) involves identifying and removing or correcting any errors or inconsistencies in the dataset. This step is important to ensure that the data is accurate and reliable.
2. **Data Transformation**
3. **Feature Extraction**
4. **Feature Selection**
    - Feature selection involves selecting the most relevant features from the dataset for use in machine learning. This can include techniques like correlation analysis, mutual information, and stepwise regression.
5. **Feature Iteration**
    - Feature iteration involves refining and improving the features based on the performance of the machine learning model. This can include techniques like adding new features, removing redundant features and transforming features in different ways.

Overall, the goal of feature engineering is to create a set of informative and relevant features that can be used to train a machine learning model and improve its accuracy and performance. The specific steps involved in the process may vary

depending on the type of data and the specific machine-learning problem at hand.

Techniques Used in Feature Engineering

Feature engineering is the process of transforming raw data into features that are suitable for machine learning models. There are various techniques that can be used in feature engineering to create new features by combining or transforming the existing ones. The following are some of the commonly used feature engineering techniques:

One-Hot Encoding

One-hot encoding is a technique used to transform categorical variables into numerical values that can be used by machine learning models. In this technique, each category is transformed into a binary value indicating its presence or absence. For example, consider a categorical variable "Colour" with three categories: Red, Green, and Blue. One-hot encoding would transform this variable into three binary variables: Colour_Red, Colour_Green, and Colour_Blue, where the value of each variable would be 1 if the corresponding category is present and 0 otherwise.

Binning

Binning is a technique used to transform continuous variables into categorical variables. In this technique, the range of values of the continuous variable is divided into several bins, and each bin is assigned a categorical value. For example, consider a continuous variable "Age" with values ranging from 18 to 80. Binning would

divide this variable into several age groups such as 18-25, 26-35, 36-50, and 51-80, and assign a categorical value to each age group.

Scaling

The most common scaling techniques are standardization and normalization. Standardization scales the variable so that it has zero mean and unit variance. Normalization scales the variable so that it has a range of values between 0 and 1.

FeatureSplit

Feature splitting is a powerful technique used in feature engineering to improve the performance of machine learning models. It involves dividing single features into multiple sub-features or groups based on specific criteria. This process unlocks valuable insights and enhances the model's ability to capture complex relationships and patterns within the data.

TextDataPreprocessing

Text data requires special preprocessing techniques before it can be used by machine learning models. Text preprocessing involves removing stop words, stemming, lemmatization, and vectorization. Stop words are common words that do not add much meaning to the text, such as "the" and "and". Stemming involves reducing words to their root form, such as converting "running" to "run". Lemmatization is similar to stemming, but it reduces words to their base form, such as converting "running" to "run". Vectorization involves transforming text data into numerical vectors that can be used by machine learning models.

Feature Engineering Tools

There are several tools available for feature engineering. Here are some popular ones:

1. Featuretools

Featuretools is a Python library that enables automatic feature engineering for structured data. It can extract features from multiple tables, including relational databases and CSV files, and generate new features based on user-defined primitives. Some of its features include:

- Automated feature engineering using machine learning algorithms.
- Support for handling time-dependent data.
- Integration with popular Python libraries, such as pandas and scikit-learn.
- Visualization tools for exploring and analyzing the generated features.
- Extensive documentation and tutorials for getting started.

2. TPOT

TPOT (Tree-based Pipeline Optimization Tool) is an automated machine learning tool that includes feature engineering as one of its components. It uses genetic programming to search for the best combination of features and machine learning algorithms for a given dataset. Some of its features include:

- Automatic feature selection and transformation.

- Support for multiple types of machine learning models, including regression, classification, and clustering.
- Ability to handle missing data and categorical variables.
- Integration with popular Python libraries, such as scikit-learn and pandas.
- Interactive visualization of the generated pipelines.

3. DataRobot

DataRobot is a machine learning automation platform that includes feature engineering as one of its capabilities. It uses automated machine learning techniques to generate new features and select the best combination of features and models for a given dataset. Some of its features include:

- Automatic feature engineering using machine learning algorithms.
- Support for handling time-dependent and text data.
- Integration with popular Python libraries, such as pandas and scikit-learn.
- Interactive visualization of the generated models and features.
- Collaboration tools for teams working on machine learning projects.

4. Alteryx

Alteryx is a data preparation and automation tool that includes feature engineering as one of its features. It provides a visual interface for creating data pipelines that can extract, transform, and generate features from multiple data sources. Some of its features include:

- Support for handling structured and unstructured data.
- Integration with popular data sources, such as Excel and databases.
- Pre-built tools for feature extraction and transformation.
- Support for custom scripting and code integration.
- Collaboration and sharing tools for teams working on data projects.

5. H2O.ai

H2O.ai is an open-source machine learning platform that includes feature engineering as one of its capabilities. It provides a range of automated feature engineering techniques, such as feature scaling, imputation, and encoding, as well as manual feature engineering capabilities for more advanced users. Some of its features include:

- Automatic and manual feature engineering options.
- Support for structured and unstructured data, including text and image data.
- Integration with popular data sources, such as CSV files and databases.
- Interactive visualization of the generated features and models.
- Collaboration and sharing tools for teams working on machine learning projects.

Overall, these tools can help streamline and automate the feature engineering process, making it easier and faster to create informative and relevant features for machine learning models

Data Representation:

Introduction:

In the realms of signal processing and machine learning, the representation of data in a numerical format is crucial for analysis, processing, and modeling. Depending on the dimensionality and complexity of the data, it can be represented in different numerical forms. The most common forms of data representation include scalar values, vectors, matrices, and tensors.

Definitions:

- **Scalar:** A scalar is a single numerical value. It has magnitude but no direction. For instance, in mathematical terms, the number 5 or -3.2 are scalar values.

- **Vector:** A vector is an ordered list of numbers. It can be visualized as a line segment in space that has both magnitude and direction. Mathematically, a vector is typically represented as a column (or sometimes row) of numbers (aka 1-D data)

- **Matrix:** A matrix is a **two-dimensional array of numbers**. It can be visualized as a rectangular grid of numbers. A matrix has rows and columns, and its shape is often described by the number of rows by the number of columns, e.g., a 3x2 matrix has 3 rows and 2 columns.

- **Tensor:** A tensor is a **multi-dimensional array of numbers**. While a scalar is 0-dimensional, a vector is 1-dimensional, and a matrix is 2-dimensional, a tensor can be 3-dimensional or more. For instance, a 3-dimensional tensor can be visualized as a cube of numbers.

Examples:

- **Scalar:** An example of a scalar is the **current temperature**. If it's 22°C right now, that's a single numerical value.

- **Vector:** An example of a vector could be the **average high temperatures forecasted for the next week**. For instance, if the forecasted high temperatures for the next seven days are 22°C, 23°C, 24°C, 25°C, 26°C, 27°C, and 28°C, then the vector representation might be: [22, 23, 24, 25, 26, 27, 28]

- **Matrix:** An example of a matrix is a **grayscale image.** The pixels of the image are represented as values between 0 (black) and 255 (white). The image's resolution, say 100x100 pixels, will determine the size of the matrix. Each entry in the matrix corresponds to the grayscale value of a pixel.

- **Tensor:** A tensor example is a **colored image**. In the most common format, an image has three color channels: Red, Green, and Blue (RGB). Each channel can be thought of as a matrix (like the grayscale image), and the three matrices combined form a 3-dimensional tensor. If the image is of resolution 100x100 pixels, the tensor's shape would be 100x100x3, with each slice of size 100x100 representing one of the RGB channels.

Summary & Conclusion

- **Scalar:** A single numerical value.

- **Vector:** A 1D array of values, often representing a series or sequence of numbers.

- **Matrix:** A 2D array of values, typically visualized as a grid or table of numbers.

- **Tensor:** An array of values with 3 or more dimensions, commonly used in applications requiring multi-channel or multi-modal data.

Data representation is crucial in domains like signal processing and machine learning. Gaining a practical understanding of how these data structures manifest in real-world engineering scenarios enables one to look beyond the terminology. Rather than being daunted by the

jargon, professionals can leverage these data formats as powerful tools for both analysis and synthesis in their respective fields.

Model Selection:

Introduction

Model selection is an essential phase in the development of powerful and precise predictive models in the field of machine learning. Model selection is the process of deciding which algorithm and model architecture is best suited for a particular task or dataset. It entails contrasting various models, assessing their efficacy, and choosing the one that most effectively addresses the issue at hand.

The choice of an appropriate machine learning model is crucial since there are various levels of complexity, underlying assumptions, and capabilities among them. A model's ability to generalize to new, untested data may not be as strong as its ability to perform effectively on a single dataset or problem. Finding a perfect balance between the complexity of models & generalization is therefore key to model selection.

Choosing a model often entails a number of processes. The first step in this process is to define a suitable evaluation metric that matches the objectives of the particular situation. According to the nature of the issue, this statistic may refer to precision, recall, accuracy, F1-score, or any other relevant measure.

The selection of numerous candidate models is then made in accordance with the problem at hand and the data that are accessible. These models might be as straightforward as decision trees or linear regression or as sophisticated as deep neural networks, random forests, or support vector machines. During the selection process, it is important to take into account the assumptions, constraints, and hyperparameters that are unique to each model.

Using a suitable methodology, such as cross-validation, the candidate models are trained and evaluated after being selected. To do this, the available data must be divided into validation and training sets, with each model fitting on the training set before being evaluated on the

validation set. The models are compared using their performance metrics, then the model with the highest performance is chosen.
Model selection is a continuous process, though. In order to make wise selections, it frequently calls for an iterative process that involves testing several models and hyperparameters. The models are improved through this iterative process, which also aids in choosing the ideal mix of algorithms & hyperparameters.

Model Selection

In machine learning, the process of selecting the top model or algorithm from a list of potential models to address a certain issue is referred to as model selection. It entails assessing and contrasting various models according to how well they function and choosing the one that reaches the highest level of accuracy or prediction power.
Because different models have varied levels of complexity, underlying assumptions, and capabilities, model selection is a crucial stage in the machine-learning pipeline. Finding a model that fits the training set of data well and generalizes well to new data is the objective. While a model that is too complex may overfit the data and be unable to generalize, a model that is too simple could underfit the data and do poorly in terms of prediction.
The following steps are frequently included in the model selection process:

- **Problem formulation:** Clearly express the issue at hand, including the kind of predictions or task that you'd like the model to carry out (for example, classification, regression, or clustering).
- **Candidate model selection:** Pick a group of models that are appropriate for the issue at hand. These models can include straightforward methods like decision trees or linear regression as well as more sophisticated ones like deep neural networks, random forests, or support vector machines.
- **Performance evaluation:** Establish measures for measuring how well each model performs. Common measurements include area under the receiver's operating characteristic curve (AUC-

ROC), recall, F1-score, mean squared error, and accuracy, precision, and recall. The type of problem and the particular requirements will determine which metrics are used.

- **Training and evaluation:** Each candidate model should be trained using a subset of the available data (the training set), and its performance should be assessed using a different subset (the validation set or via cross-validation). The established evaluation measures are used to gauge the model's effectiveness.
- **Model comparison:** Evaluate the performance of various models and determine which one performs best on the validation set. Take into account elements like data handling capabilities, interpretability, computational difficulty, and accuracy.
- **Hyperparameter tuning:** Before training, many models require that certain hyperparameters, such as the learning rate, regularisation strength, or the number of layers that are hidden in a neural network, be configured. Use methods like grid search, random search, and Bayesian optimization to identify these hyperparameters' ideal values.
- **Final model selection:** After the models have been analyzed and fine-tuned, pick the model that performs the best. Then, this model can be used to make predictions based on fresh, unforeseen data.

Model Selection in machine learning:

Model selection in machine learning is the process of selecting the best algorithm and model architecture for a specific job or dataset. It entails assessing and contrasting various models to identify the one that best fits the data & produces the best results. Model complexity, data handling capabilities, and generalizability to new examples are all taken into account while choosing a model. Models are evaluated and contrasted using methods like cross-validation, and grid search, as well as indicators like accuracy and mean squared error. Finding a model that balances complexity and performance to produce reliable predictions and strong generalization abilities is the aim of model selection.

There are numerous important considerations to bear in mind while selecting a model for machine learning. These factors assist in

ensuring that the chosen model is effective in solving the issue at its core and has an opportunity for outstanding performance. Here are some crucial things to remember:

- **The complexity of the issue:** Determine how complex the issue you're trying to resolve is. Simple models might effectively solve some issues, but more complicated models can be necessary to fully represent complex relationships in the data. Take into account the size of the dataset, the complexity of the input features, and any potential for non-linear connections.
- **Data Availability & Quality:** Consider the accessibility and caliber of the data you already have. Using complicated models with a lot of parameters on a limited dataset may result in overfitting. Such situations may call for simpler models with fewer parameters. Take into account missing data, outliers, and noise as well as how various models respond to these difficulties.
- **Interpretability:** Consider whether the model's interpretability is crucial in your particular setting. Some models, like decision trees or linear regression, offer interpretability by giving precise insights into the correlations between the input data and the desired outcome. Complex models, such as neural networks, may perform better but offer less interpretability.
- **Model Assumptions:** Recognise the presumptions that various models make. For instance, although decision trees assume piecewise constant relationships, linear regression assumes a linear relationship between the input characteristics and the target variable. Make sure the model you choose is consistent with the fundamental presumptions underpinning the data and the issue.
- **Scalability and Efficiency:** If you're working with massive datasets or real-time applications, take the model's scalability and computing efficiency into consideration. Deep neural networks and support vector machines are two examples of models that could need a lot of time and computing power to train.
- **Regularisation and Generalisation:** Assess the model's capacity to apply to fresh, untested data. By including penalty terms to the objective function of the model, regularisation

approaches like L1 or L2 regularisation can help prevent overfitting. When the training data is sparse, regularised models may perform better in terms of generalization.

- **Domain Expertise:** Consider your expertise and domain knowledge. On the basis of previous knowledge of the data or particular features of the domain, consider if particular models are appropriate for the task. Models that are more likely to capture important patterns can be found by using domain expertise to direct the selection process.
- **Resource Constraints:** Take into account any resource limitations you may have, such as constrained memory space, processing speed, or time. Make that the chosen model can be successfully implemented using the resources at hand. Some models require significant resources during training or inference.
- **Ensemble Methods:** Examine the potential advantages of ensemble methods, which integrate the results of various models in order to perform more effectively. By utilizing the diversity of several models' predictions, ensemble approaches, such as bagging, boosting, and stacking, frequently outperform individual models.
- **Evaluation and Experimentation:** experimentation and assessment of several models should be done thoroughly. Utilize the right evaluation criteria and statistical tests to compare their performance. To evaluate the models' performance on unknown data and reduce the danger of overfitting, use hold-out or cross-validation.

Model Selection Techniques

Model selection in machine learning can be done using a variety of methods and tactics. These methods assist in comparing and assessing many models to determine which is best suited to solve a certain issue. Here are some methods for selecting models that are frequently used:

- **Train-Test Split:** With this strategy, the available data is divided into two sets: a training set & a separate test set. The models are evaluated using a predetermined evaluation metric on the test set after being trained on the training set. This

method offers a quick and easy way to evaluate a model's performance using hypothetical data.

- **Cross-Validation:** A resampling approach called cross-validation divides the data into various groups or folds. Several folds are used as the test set & the rest folds as the training set, and the models undergo training and evaluation on each fold separately. Lowering the variance in the evaluation makes it easier to generate an accurate assessment of the model's performance. Cross-validation techniques that are frequently used include leave-one-out, stratified, and k-fold cross-validation.
- **Grid Search:** Hyperparameter tuning is done using the grid search technique. In order to do this, a grid containing hyperparameter values must be defined, and all potential hyperparameter combinations must be thoroughly searched. For each combination, the models are trained, assessed, and their performances are contrasted. Finding the ideal hyperparameter settings to optimize the model's performance is made easier by grid search.
- **Random Search:** A set distribution for hyperparameter values is sampled at random as part of the random search hyperparameter tuning technique. In contrast to grid search, which considers every potential combination, random search only investigates a portion of the hyperparameter field. When a thorough search is not possible due to the size of the search space, this strategy can be helpful.
- **Bayesian optimization:** A more sophisticated method of hyperparameter tweaking, Bayesian optimization. It models the relationship between the performance of the model and the hyperparameters using a probabilistic model. It intelligently chooses which set of hyperparameters to investigate next by updating the probabilistic model and iteratively assessing the model's performance. When the search space is big and expensive to examine, Bayesian optimization is especially effective.
- **Model averaging:** This technique combines forecasts from various models to get a single prediction. For regression issues, this can be accomplished by averaging the predictions, while for

classification problems, voting or weighted voting systems can be used. Model averaging can increase overall prediction accuracy by lowering the bias and variation of individual models.

- **Information Criteria:** Information criteria offer a numerical assessment of the trade-off between model complexity and goodness of fit. Examples include the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). These criteria discourage the use of too complicated models and encourage the adoption of simpler models that adequately explain the data.
- **Domain Expertise & Prior Knowledge:** Prior understanding of the problem and the data, as well as domain expertise, can have a significant impact on model choice. The models that are more suitable given the specifics of the problem and the details of the data may be known by subject matter experts.
- **Model Performance Comparison:** Using the right assessment measures, it is vital to evaluate the performance of various models. Depending on the issue at hand, these measurements could include F1-score, mean squared error, accuracy, precision, recall, or the area beneath the receiver's operating characteristic curve (AUC-ROC). The best-performing model can be found by comparing many models.

Summary

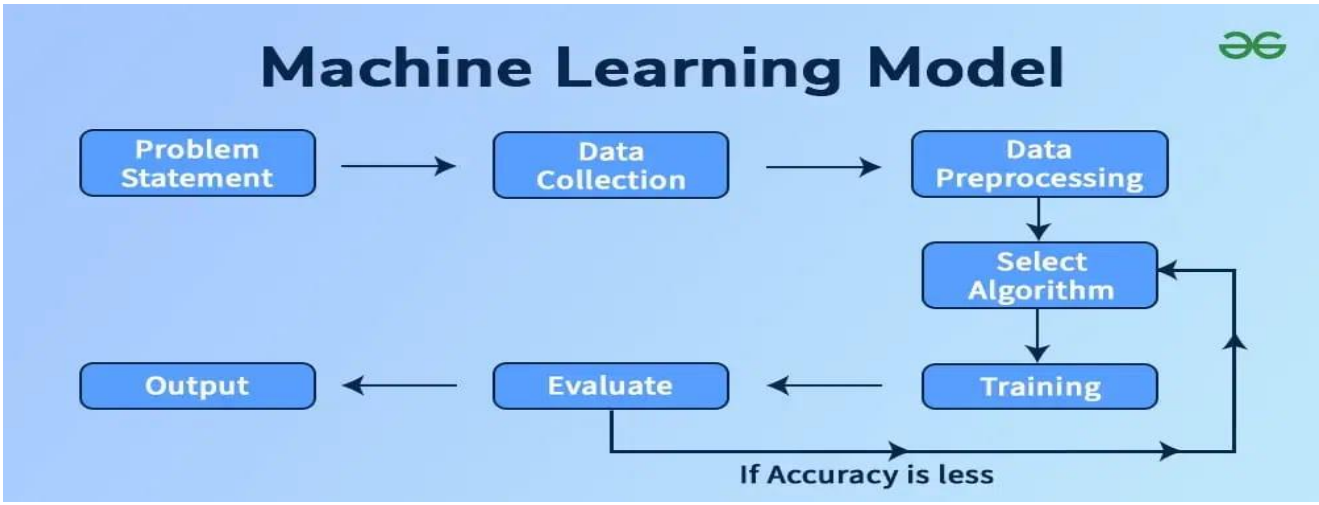
The important machine learning stage of model selection entails selecting the best model and algorithm for a certain task. To make precise predictions on unknown data, it is crucial to find a balance between model complexity & generalization. Model selection involves selecting potential candidates, assessing each model's performance, and selecting the model with the best results.

Assessing the problem's complexity, data quality and availability, interpretability, model assumptions, scalability, efficiency, regularisation, domain knowledge, resource restrictions, and the possible advantages of ensemble approaches are all factors that should be taken into account when choosing a model. These factors aid in

ensuring that the chosen model complies with the limits and needs of the issue.
There are many methods for choosing a model, such as train-test split, cross-validation, grid searches, random search, Bayesian optimization, model averaging, information criteria, expertise in the domain, and model performance comparison. These methods make it possible to thoroughly assess, tune hyperparameters, and compare various models to get the best fit.

Model Learning:

Machine learning models are very power powerful resources that automate tasks and make them more accurate and efficient.ML handles new data and scales the growing demand for technology with valuable insight .It improves the performance over time .This cutting-edge technology has various benefits such as faster processing or response ,enhancement of decision-making,and specialized services.



A model of machine learning is a set of programs that can be used to find the pattern and make a decision from an unseen dataset .These days NLP[Natural language processing] uses the machine learning model to recognize the unstructured text into usable data and insights.You may have heared about image processing which is used to

identify objects such as boy,girl,mirror,car,dog,etc.A model always requires a dataset to perform various tasks during training duration,We use a machine learning algorithm for the optimizing process to find certain patterns or outputs from the dataset based upon tasks

Types of Machine Learning Models

Machine learning models can be brodly categorized into four main paradigms based on the type of data and learning goals:

1. Supervised Models

Supervised learning is the study of algorithms that use labelled data in which each data instance has a known category or value to which it belongs.This results in the model to disciver the relationship between the input features and the target outcome.

1.1 Classification

The classifier algorithms are designed to indicate whether a new data point belongs to one or another among several predefined classes. Imagine when you are organising emails into spam or inbox, categorising images as cat or dog, or predicting whether a loan applicant is a credible borrower. In the classification models, there is a learning process by the use of labeled examples from each category. In this process, they discover the correlations and relations within the data that help to distinguish class one from the other classes. After learning these patterns, the model is then capable of assigning these class labels to unseen data points.

Common Classification Algorithms:

- **Logistic Regression:** A very efficient technique for the classification problems of binary nature (two types, for example, spam/not spam).
- **Support Vector Machine (SVM):** Good for tasks like classification, especially when the data has a large number of features.

- **Decision Tree:** Constructs a decision tree having branches and proceeds to the class predictions through features.
- **Random Forest:** The model generates an "ensemble" of decision trees that ultimately raise the accuracy and avoid overfitting (meaning that the model performs great on the training data but lousily on unseen data).
- **K-Nearest Neighbors (KNN):** Assigns a label of the nearest neighbors for a given data point.

1.2 Regression

Regression algorithms are about forecasting of a continuous output variable using the input features as their basis. This value could be anything such as predicting real estate prices or stock market trends to anticipating customer churn (how likely customers stay) and sales forecasting. Regression models make the use of features to understand the relationship among the continuous features and the output variable. That is, they use the pattern that is learned to determine the value of the new data points.

Common Regression Algorithms

- **Linear Regression:** Fits depth of a line to the data to model for the relationship between features and the continuous output.
- **Polynomial Regression:** Similiar to linear regression but uses more complex polynomial functions such as quadratic, cubic, etc, for accommodating non-linear relationships of the data.
- **Decision Tree Regression:** Implements a decision tree-based algorithm that predicts a continuous output variable from a number of branching decisions.
- **Random Forest Regression:** Creates one from several decision trees to guarantee error-free and robust regression prediction results.
- **Support Vector Regression (SVR):** Adjusts the Support Vector Machine ideas for regression tasks, where we are trying to find one hyperplane that most closely reflects continuous output data.

2. Unsupervised Models

Unsupervised learning involves a difficult task of working with data which is not provided with pre-defined categories or label.

2.1 **Clustering**

Visualize being given a basket of fruits with no labels on them. The fruits clustering algorithms are to group them according to the inbuilt similarities. Techniques like K-means clustering are defined by exact number of clusters ("red fruits" and "green fruits") and then each data point (fruit) is assigned to the cluster with the highest similarity within based on features (color, size, texture). Contrary to this, hierarchical clustering features construction of hierarchy of clusters which makes it more easy to study the system of groups. Spatial clustering algorithm Density-Based Spatial Clustering of Applications with Noise (DBSCAN) detects groups of high-density data points, even in those areas where there is a lack of data or outliers.

2.2 **Dimensionality Reduction**

Sometimes it is difficult to both visualize and analyze the data when you have a large feature space (dimensions). The purpose of dimensionality reduction methods is to decrease the dimensions needed to maintain the key features. Dimensions of greatest importance are identified by principal component analysis (PCA), which is the reason why data is concentrated in fewer dimensions with the highest variations. This speeds up model training as well as offers a chance for more efficient visualization. LDA (Linear Discriminant Analysis) also resembles PCA but it is made for classification tasks where it concentrates on dimensions that can differentiate the present classes in the dataset.

2.3 **Anomaly Detection**

Unsupervised learning can also be applied to find those data points which greatly differ than the majorities. The statistics model may identify these outliers, or anomalies as signaling of errors, fraud or even something unusual. Local Outlier Factor (LOF) makes a comparison of a given data point's local density with those surrounding it. It then flags out the data points with significantly lower densities as outliers or potential anomalies. Isolation Forest is the one which uses different approach, which is to recursively isolate data points according to their features. Anomalies usually are simple to contemplate as they often necessitate fewer steps than an average normal point.

3. Semi-Supervised Model

Besides, supervised learning is such a kind of learning with labeled data that unsupervised learning, on the other hand, solves the task where there is no labeled data. Lastly, semi-supervised learning fills the gap between the two. It reveals the strengths of both approaches by training using data sets labeled along with unlabeled one. This is especially the case when labeled data might be sparse or prohibitively expensive to acquire, while unlabeled data is undoubtedly available in abundance.

3.1 Generative Semi-Supervised Learning

Envision having a few pictures of cats with labels and a universe of unlabeled photos. The big advantage of generative semi-supervised learning is its utilization of such a scenario. It exploits a generative model to investigate the unlabeled pictures and discover the orchestrating factors that characterize the data. This technique can then be used to generate the new synthetic data points that have the same features with the unlabeled data. The synthetic data is then labeled with the pseudo-labels that the generative model has interpreted from the data. This approach combines the existing labeled data with the newly generated labeled data to train the final model which is likely to perform better than the previous model that was trained with only the limited amount of the original labeled data.

3.2 Graph-based Semi-Supervised Learning

This process makes use of the relationships between data points and propagates labels to unmarked ones via labeled ones. Picture a social network platform where some of the users have been marked as fans of sports (labeled data). Cluster-based methods can analyze the links between users (friendships) and even apply this information to infer that if a user is connected to someone with a "sports" label then this user might also be interested in sports (unbiased labels with propagated label). While links and the entire structure of the network are also important for the distribution of labels. This method is beneficial when the data points are themselves connected to each other and this connection can be exploiting during labelling of new data.

4. Reinforcement learning Models

Reinforcement learning takes a dissimilar approach from supervised learning and unsupervised learning. Different from supervised

learning or just plain discovery of hidden patterns, reinforcement learning adopt an agent as it interacts with the surrounding and learns. This agent is a learning one which develops via experiment and error, getting rewarded for the desired actions and punished for the undesired ones. The main purpose is to help players play the game that can result in the highest rewards.

4.1 Value-based learning:

Visualize a robot trying to find its way through a maze. It has neither a map nor instructions, but it gets points for consuming the cheese at the end and fails with deduction of time when it runs into a wall. Value learning is an offshoot of predicting the anticipated future reward of taking a step in a particular state. For example, the algorithm Q-learning will learn a Q-value for each state-action combination. This Q-value is the expected reward for that action at that specific state. Through a repetitive process of assessing the state, gaining rewards, and updating the Q-values the agent manages to determine that which actions are most valuable in each state and eventually guides it to the most rewarding path. In contrast, SARSA (State-Action-Reward-State-Action) looks at the value of the succeeding state-action pair that influences the exploration strategy.

4.2 Policy-based learning:

In contrast to the value-based learning, where we are learning a specific value for each state-action pair, in policy-based learning we are trying to directly learn a policy which maps states to actions. This policy in essence commands the agent to act in different situations as specified by the way it is written. Actor-Critic is a common approach that combines two models: an actor that retrains the policy and a critic that retrains the value function (just like value-based methods). The actor witnesses the critic's feedback which updates the policy that the actor uses for better decision making. Proximal Policy Optimization (PPO) is a specific policy-based method which focuses on high variance issues that complicate early policy-based learning methods.

Deep Learning

Deep learning is a subfield of machine learning that utilizes artificial neural networks with multiple layers to achieve complex pattern recognition. These networks are particularly effective for tasks

involving large amounts of data, such as image recognition and natural language processing.

1. **Artificial Neural Networks (ANNs)** - This is a popular model that refers to the structure and function of the human brain. It consists of interconnected nodes based on various layers and is used for various ML tasks.
2. **Convolutional Neural Networks (CNNs) -** A CNN is a deep learning model that automates the spatial hierarchies of features from input data. This model is commonly used in image recognition and classification.
3. **Recurrent Neural Networks (RNNs) -** This model is designed for the processing of sequential data. It enables the memory input which is known for Neural network architectures.
4. **Long Short-Term Memory Networks (LSTMs) -** This model is comparatively similar to Recurrent Neural Networks and allows learners to learn the long-term dependencies from sequential data.

How Machine Learning Works?

1. **Model Represntation:** Machine Learning Models are represented by mathematical functions that map input data to output predictions. These functions can take various forms, such as linear equations, decision trees , or complex neural networks.
2. **Learning Algorithm:** The learning algorithm is the main part of behind the model's ability to learn from data. It adjusts the parameters of the model's mathematical function iteratively during the training phase to minimize the difference between the model's prediction and the actual outcomes in the training data .
3. **Training Data:** Training data is used to teach the model to make accurate predictions. It consists of input features(e.g variables, attributes) and corresponding output labels(in supervised learning) or is unalabeled(in supervised learning). During training , the model analyzes the patterns in the training data to update its parameters accordingly.
4. **Objective Function:** The objective function, also known as the loss function, measures the difference between the model's predictions and the actual outcomes in the training data. The goal during training is to minimize this function, effectively reducing the errors in the model's predictions.

5. **Optimization Process:** Optimization is the process of finding the set of model parameters that minimize the objective function. This is typically achieved using optimization algorithms such as gradient descent, which iteratively adjusts the model's parameters in the direction that reduces the objective function.
6. **Generalization:** Once the model is trained, it is evaluated on a separate set of data called the validation or test set to assess its performance on new, unseen data. The model's ability to perform well on data it hasn't seen before is known as generalization.
7. **Final Output:** After training and validation, the model can be used to make predictions or decisions on new, unseen data. This process, known as inference, involves applying the trained model to new input data to generate predictions or classifications.

Advanced Machine Learning Models

- **Neural Networks**: You must have heard about deep neural network which helps solve complex problems of data. It is made up of interconnected nodes of multiple layers which we also call neurons. Many things have been successful from this model such as image recognition, NLP, and speech recognition.
- **Convolutional Neural Networks (CNNs)**: This is a type of model that is built in the framework of a neural network and it is made to handle data that are of symbolic type, like images. From this model, the hierarchy of spatial features can be determined.
- **Recurrent Neural Networks (RNNs)**: These can be used to process data that is sequentially ordered, such as reading categories or critical language. These networks are built with loops in their architectures that allow them to store information over time.
- **Long Short-Term Memory Networks (LSTMs)**: LSTMs, which are a type of RNNs, recognize long-term correlation objects. These models do a good job of incorporating information organized into long categories.
- **Generative Adversarial Networks (GANs)**: GANs are a type of neural networks that generate data by studying two networks over time. A product generates network data, while a determination attempts to distinguish between real and fake samples.

- **Transformer Models**: This model become popular in natural language processing. These models process input data over time and capture long-range dependencies.

Real-world examples of ML Models

The ML model uses predictive analysis to maintain the growth of various Industries-

- **Financial Services**: Banks and financial institutions are using machine learning models to provide better services to their customers. Using intelligent algorithms, they understand customers' investment preferences, speed up the loan approval process, and receive alerts for non-ordinary transactions.
- **Healthcare**: In medicine, ML models are helpful in disease prediction, treatment recommendations, and prognosis. For example, physicians can use a machine learning model to predict the right cold medicine for a patient.
- **Manufacturing Industry**: In the manufacturing sector, ML has made the production process more smooth and optimized. For example, Machine Learning is being used in automated production lines to increase production efficiency and ensure manufacturing quality.
- **Commercial Sector**: In the marketing and marketing sector, ML models analyze huge data and predict production trends. This helps in understanding the marketing system and the products can be customized for their target customers.

Future of Machine Learning Models

There are several important aspects to consider when considering the challenges and future of machine learning models. One challenge is that there are not enough resources and tools available to contextualize large data sets. Additionally, machine learning models need to be updated and restarted to understand new data patterns.

In the future, another challenge for machine learning may be to collect and aggregate collections of data between different existing technology versions. This can be important for scientific development along with promoting the discovery of new possibilities. Finally, good strategy, proper resources, and technological advancement are important concepts for success in developing machine learning models. To address all these challenges, appropriate time and attention is required to further expand machine learning capabilities.

Conclusion

We first saw the introduction of machine learning in which we know what a model is and what is the benefit of implementing it in our system. Then look at the history and evolution of machine learning along with the selection criteria to decide which model to use specifically. Next, we read data preparation where you can read all the steps. Then we researched advanced model that has future benefits but some challenges can also be faced but the ML model is a demand for the future.

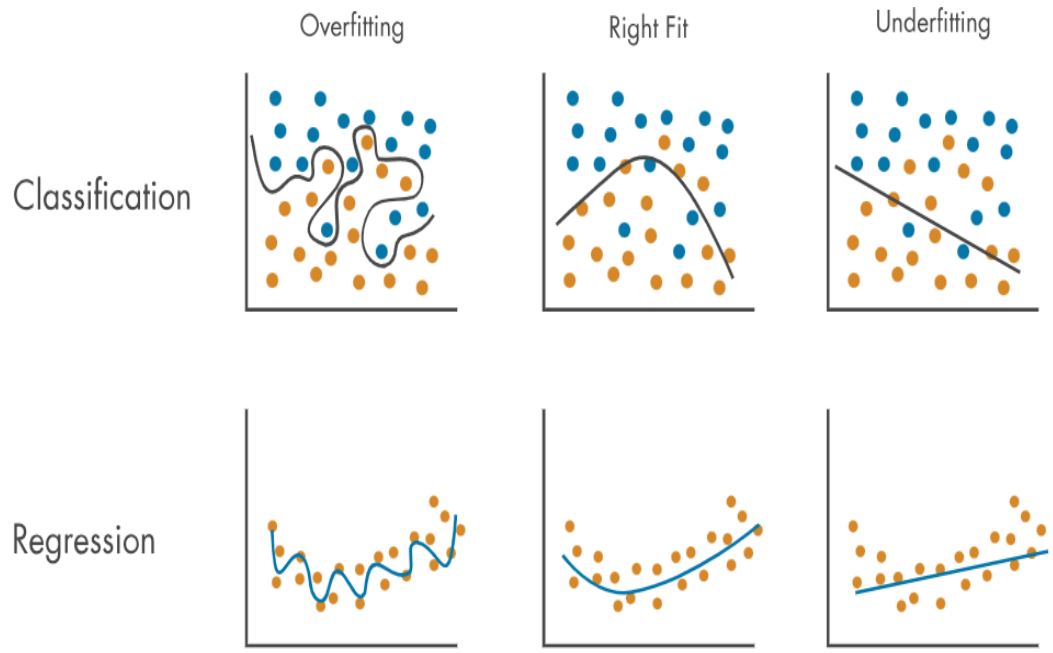
Model evaluation:

What is Model Evaluation?

Model evaluation is the process of using different evaluation metrics to understand a machine learning model's performance, as well as its strengths and weaknesses.

Why is Evaluation necessary for a successful model?

Evaluation is necessary for ensuring that machine learning models are reliable, generalizable, and capable of making accurate predictions on new, unseen data, which is crucial for their successful deployment in real-world applications. Overfitting and underfitting are the two biggest causes of poor performance of machine learning algorithms.



Overfitting: *Occurs when the model is* ***so closely*** *aligned to the training data that it does not know how to respond to new data.*

Underfitting*: Occurs when the model* ***cannot adequately capture*** *the underlying structure of the data.*

Right Fit: *Occurs when both the training data error and the test data are minimal*

Error	Overfitting	Right Fit	Underfitting
Training	Low	Low	High
Test	High	Low	High

Error Risks in the models

Evaluation Metrics

There are different metrics for the tasks of classification, regression, ranking, clustering, topic modeling, etc. Some of the metrics are as follows:

1. *Classification Metrics (accuracy, precision, recall, F1-score, ROC, AUC, ...)*
2. *Regression Metrics (MSE, MAE, R2)*
3. *Ranking Metrics (MRR, DCG, NDCG)*
4. *Statistical Metrics (Correlation)*
5. *Computer Vision Metrics (PSNR, SSIM, IoU)*
6. *NLP Metrics (Perplexity, BLEU score)*
7. *Deep Learning Related Metrics (Inception score, Frechet Inception distance)*

→ *Today, we will talk about Classification Metrics.*

1. Classification Metrics

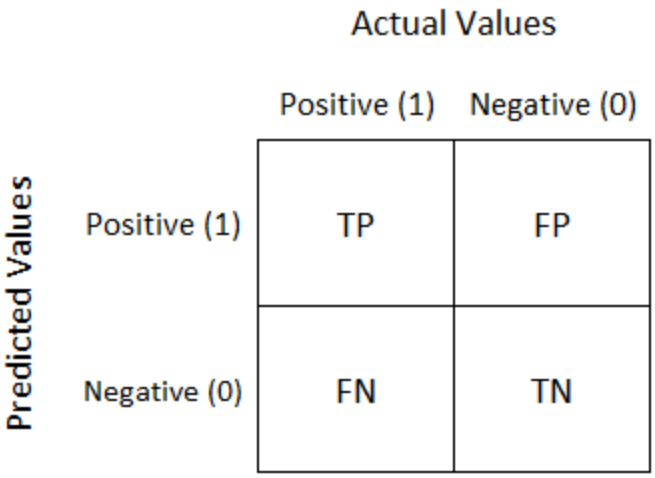
When our target is categorical, we are dealing with a classification problem. The choice of the most appropriate metrics depends on different aspects, such as the characteristics of the dataset, whether it's imbalanced or not, and the goals of the analysis.

Confusion Matrix

A confusion matrix is a table that is often used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known.

I can summarize as before and after happenings. How?

As you see we have 2 main situations. Predicted (Before), Actual Values (After).

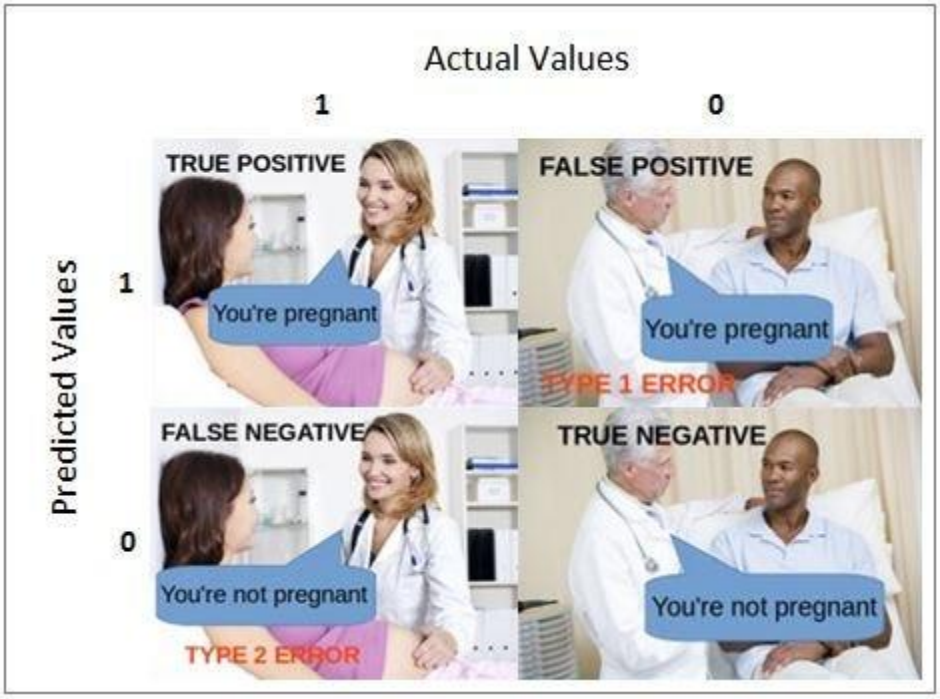


Predicted and Actual Values

1. ***Predicted*:** Negative & ***Actual Value***: Positive → Your predicted False (FN)
2. ***Predicted*:** Negative & ***Actual Value***: Negative → Your predicted True(TN)
3. ***Predicted*:** Pozitive & ***Actual Value***: Positive → Your predicted True (TP)

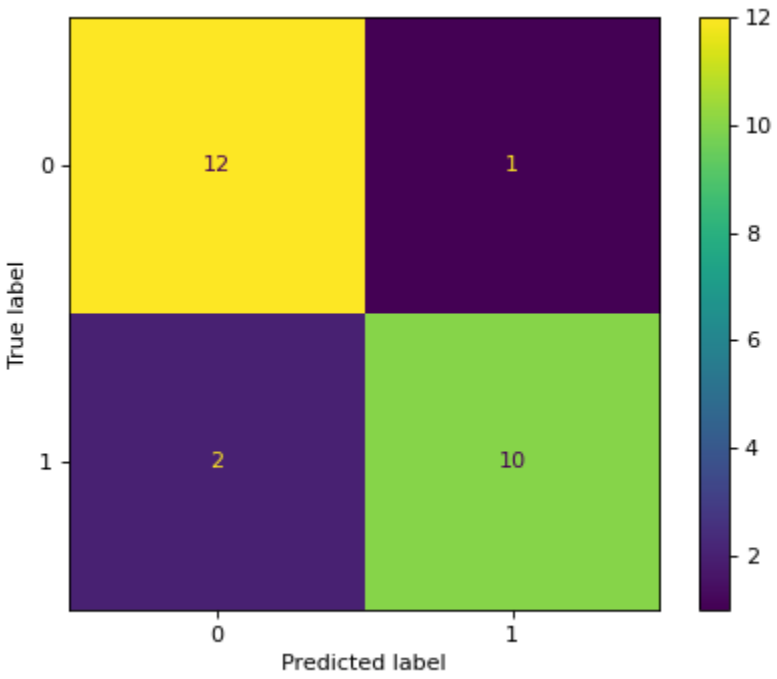
4. ***Predicted*:** Pozitive & ***Actual Value***: Negative→ Your predicted False (FP)

These four scenarios are illustrated in the following figure.



Four possible combinations of reality and our binary pregnancy test results

Example Label for Accuracy, Precision, and Recall



True Positive (TP) =10

True Negative (TN)=12

False Positive (FP)=1

False Negative (FN)=2

Accuracy

Accuracy is one metric for evaluating classification models. Formally accuracy could be defined as the number of correct predictions to a total number of predictions.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

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

$$Accuracy = \frac{10 + 12}{10 + 12 + 1 + 2} = 88\%$$

Precision

Precision is a measure of the accuracy.

$$\textbf{Precision} = \frac{\textbf{TP}}{\textbf{TP + FP}}$$

$$\text{Precision} = \frac{10}{10 + 1} = 91\%$$

Recall

Recall is the true positive rate

$$\textbf{Recall} = \frac{\textbf{TP}}{\textbf{TP + FN}}$$

$$Recall = \frac{10}{10 + 2} = 83\%$$

F1 Score

F1 score is a machine learning evaluation metric that measures a model's accuracy. It combines the precision and recall scores of a model.

The accuracy metric computes how many times a model made a correct prediction across the entire dataset.

$$\mathbf{F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}}$$

$$F1 = \frac{2 * 0.91 * 0.83}{0.91 + 0.83} = 0.87$$

In some scenarios, precision and recall may have varying levels of importance depending on the specific requirements of the application. The F1 score, which balances both precision and recall, may not perfectly capture the relative importance of these metrics for a given task. F1 score or seeing the PR or ROC curve can help.

ROC

ROC curve provides a comprehensive view of a model's ability to discriminate between classes, especially in binary classification tasks. It helps in understanding the trade-offs between sensitivity and

specificity at different decision thresholds, and the AUC offers a single metric for summarizing the overall performance of the model.

- True Positive Rate (Recall)
- False Positive Rate (FPR)

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

$$FPR = \frac{1}{1 + 12} = 0.077$$

Model prediction:

What Is Predictive Modeling?

In short, predictive modeling is a statistical technique using machine learning and data mining to predict and forecast likely future outcomes with the aid of historical and existing data. It works by analyzing current and historical data and projecting what it learns on a model generated to forecast likely outcomes. Predictive modeling can be used to predict just about anything, from TV ratings and a customer's next purchase to credit risks and corporate earnings.

A predictive model is not fixed; it is validated or revised regularly to incorporate changes in the underlying data. In other words, it's not a one-and-done prediction. Predictive models make assumptions based

on what has happened in the past and what is happening now. If incoming, new data shows changes in what is happening now, the impact on the likely future outcome must be recalculated, too. For example, a software company could model historical sales data against marketing expenditures across multiple regions to create a model for future revenue based on the impact of the marketing spend.

Most predictive models work fast and often complete their calculations in real time. That's why banks and retailers can, for example, calculate the risk of an online mortgage or credit card application and accept or decline the request almost instantly based on that prediction.

Some predictive models are more complex, such as those used in computational biology and quantum computing; the resulting outputs take longer to compute than a credit card application but are done much more quickly than was possible in the past thanks to advances in technological capabilities, including computing power.

Top 5 Types of Predictive Models

Fortunately, predictive models don't have to be created from scratch for every application. Predictive analytics tools use a variety of vetted models and algorithms that can be applied to a wide spread of use cases.

Predictive modeling techniques have been perfected over time. As we add more data, more muscular computing, AI and machine learning and see overall advancements in analytics, we're able to do more with these models.

The top five predictive analytics models are:

1. **Classification model:** Considered the simplest model, it categorizes data for simple and direct query response. An

example use case would be to answer the question “Is this a fraudulent transaction?”

2. **Clustering model:** This model nests data together by common attributes. It works by grouping things or people with shared characteristics or behaviors and plans strategies for each group at a larger scale. An example is in determining credit risk for a loan applicant based on what other people in the same or a similar situation did in the past.
3. **Forecast model:** This is a very popular model, and it works on anything with a numerical value based on learning from historical data. For example, in answering how much lettuce a restaurant should order next week or how many calls a customer support agent should be able to handle per day or week, the system looks back to historical data.
4. **Outliers model:** This model works by analyzing abnormal or outlying data points. For example, a bank might use an outlier model to identify fraud by asking whether a transaction is outside of the customer’s normal buying habits or whether an expense in a given category is normal or not. For example, a $1,000 credit card charge for a washer and dryer in the cardholder’s preferred big box store would not be alarming, but $1,000 spent on designer clothing in a location where the customer has never charged other items might be indicative of a breached account.
5. **Time series model:** This model evaluates a sequence of data points based on time. For example, the number of stroke patients admitted to the hospital in the last four months is used to predict how many patients the hospital might expect to admit next week, next month or the rest of the year. A single metric measured and compared over time is thus more meaningful than a simple average.

Common Predictive Algorithms

Predictive algorithms use one of two things: machine learning or deep learning. Both are subsets of artificial intelligence (AI). Machine learning (ML) involves structured data, such as spreadsheet or machine data. Deep learning (DL) deals with unstructured data such as video, audio, text, social media posts and images—essentially the stuff that humans communicate with that are not numbers or metric reads.

Some of the more common predictive algorithms are:

1. **Random Forest:** This algorithm is derived from a combination of decision trees, none of which are related, and can use both classification and regression to classify vast amounts of data.
2. **Generalized Linear Model (GLM) for Two Values:** This algorithm narrows down the list of variables to find "best fit." It can work out tipping points and change data capture and other influences, such as categorical predictors, to determine the "best fit" outcome, thereby overcoming drawbacks in other models, such as a regular linear regression.
3. **Gradient Boosted Model:** This algorithm also uses several combined decision trees, but unlike Random Forest, the trees are related. It builds out one tree at a time, thus enabling the next tree to correct flaws in the previous tree. It's often used in rankings, such as on search engine outputs.
4. **K-Means:** A popular and fast algorithm, K-Means groups data points by similarities and so is often used for the clustering model. It can quickly render things like personalized retail offers to individuals within a huge group, such as a million or more customers with a similar liking of lined red wool coats.
5. **Prophet:** This algorithm is used in time-series or forecast models for capacity planning, such as for inventory needs, sales quotas and resource allocations. It is highly flexible and can easily accommodate heuristics and an array of useful assumptions.

Search and learning:

Whenever someone performs a search, they expect to be greeted with results relevant to their requirements. However, traditional search techniques like "BM25 retrieval" return results based on how many times the phrase searched appears in a document.

While such techniques perform well to an extent, they fail to take into account the users' unique preferences. This is where machine learning (ML) techniques come into play for helping deliver personalized results, particularly within the realm of federated search.

Maximizing Federated Search Relevance with ML Techniques

Unlike conventional methods, ML-driven federated search delves deeper into the intricacies of user interactions, considering many factors beyond keyword frequency.

This enables the system to discern patterns, user intent, and contextual relevance, therefore delivering a more personalized and tailored search experience. The following ML techniques help facilitate this:

Vector and Semantic Search

Vector search is a technique that leverages mathematical representations to understand and organize complex relationships between words and concepts, enhancing search accuracy. While semantic search focuses on interpreting the meaning of words and phrases within the context, providing a nuanced understanding of the users' context.

These approaches excel in handling synonyms, misspellings, and variations in language, contributing to a higher level of search

accuracy. They also boost personalization based on the users' history and preferences, something which we could see lacking in the traditional approaches.

Query Understanding

Query Understanding involves analyzing user queries to grasp their intent, context, and semantics. It employs natural language processing (NLP) to interpret user input, discerning synonyms, and user-specific language, thus enhancing search engines' ability to deliver more accurate and relevant results.

Using ML algorithms also enables the search engines to "Query rephrasing," which essentially refers to suggesting alternate queries that would retrieve better results. Another technique, known as "Query expansion" helps expand the query to add related terms to the query and broadens its scope.

And voila! The search results powered by query understanding are much more thorough.

Neural Reranking

"Ranking" is often done based on how many times the search keyword appears in a document. However, neural reranking allows you to tune the search results so the top result is the most relevant one instead.

This is done by leveraging algorithms like k-nearest-neighbor (k-NN) for exact matches and approximate nearest-neighbor (ANN) for faster but slightly less accurate matches. Another benefit of neural reranking is that it can yield great results in zero-shot informational retrieval (IR) models ,i.e., models without much training.

Now if you were to implement these ML-based techniques for better search results, how would you determine if they're performing as expected? By finding out their impact on recall and precision. Let's dig deeper.

How do ML Techniques Enhance Recall & Precision in Search Results?

Precision refers to how accurate the search results are, while recall refers to the number of results returned. ML algorithms can ensure that the no relevant results are skipped over, and simultaneously rank the most relevant matches higher to reduce irrelevancy.

The following ML techniques help boost precision and recall:

- **Word Embeddings**: Embeddings are dense vector representations that capture semantic relationships and similarities between data points. They boost recall by capturing nuanced similarities in the data, allowing models to return better results.
- **Cross-Encoders:** These are neural networks that analyze pairs of things, such as questions and answers to determine how similar they are. The model gives a score to show how much the two things are connected or similar. This helps boost precision.
- **User History:** Taking the users' history into account helps search engines narrow what they might be looking for. This historical data can be used to fine-tune the search results and provide more personalized and relevant answers.

Therefore, integrating ML techniques into your federated search solutions is a great way to amp up search relevancy. SearchUnify has been at it for quite some time! Keep reading to know more.

Delivering Search Excellence with SearchUnify's ML Techniques

SearchUnify uses three types of ML-powered search techniques to boost **recall** as explained below.

Lexical Search

Lexical refers to the vocabulary or words used in a language, such as their structure and meaning. Lexical search involves searching for specific words or terms within a dataset or a corpus of text. It focuses on finding documents or information that contain the exact words or phrases specified in the search query.

Neural Search

Neural search leverages ML models known as neural networks to improve the efficiency and relevance of search results. It leverages advanced NLP techniques to understand the context, semantics, and relationships between words. This helps provide more accurate and contextually relevant search results by understanding the meaning behind the queries.

Hybrid Search

Hybrid search combines multiple search approaches or technologies to enhance the overall search experience. It often involves integrating traditional search methods with newer technologies like ML or artificial intelligence (AI).

For example, it might use lexical search for precise keyword matching and neural search for understanding context and providing more nuanced results. Search Unify uses this combination to provide a more comprehensive and accurate search experience.

To improve **precision**, Search Unify federated search leverages the following techniques.

Auto Boosting

This helps optimize search precision by adjusting the importance of different features, ensuring relevant information is prioritized based on user interactions and feedback.

Persona-based Results

ML algorithms can track and take into account the user personas and tailor the results to user-specific profiles, considering individual preferences and behavior to deliver more accurate and personalized information.

Cross Encoders

Cross encoders utilize neural networks to analyze relationships between different pieces of content, facilitating a deeper understanding of context and relevance for more accurate search results.

Data Set:

Mahine Learning is at the peak of its popularity today. Despite this, a lot of decision-makers are in the dark about what exactly is needed to design, train, and successfully deploy a machine learning algorithm. The details about collecting the data, building a dataset, and annotation specifics are neglected as supportive tasks.

However, reality shows that working with datasets is the most time-consuming and laborious part of any AI project, sometimes taking up to 70% of the time overall. Moreover, building up a high-quality

machine learning dataset requires experienced, trained professionals who know what to do with the actual data that can be collected.

Let's start from the beginning by defining what a dataset for machine learning is and why you need to pay more attention to it.

What Is a Dataset in Machine Learning and Why Is It Essential for Your AI Model?

According to the Oxford Dictionary, a dataset definition in machine learning is "a collection of data that is treated as a single unit by a computer". This means that a dataset contains a lot of separate pieces of data, but can be used to teach the machine learning algorithm to find predictable patterns inside the whole dataset.

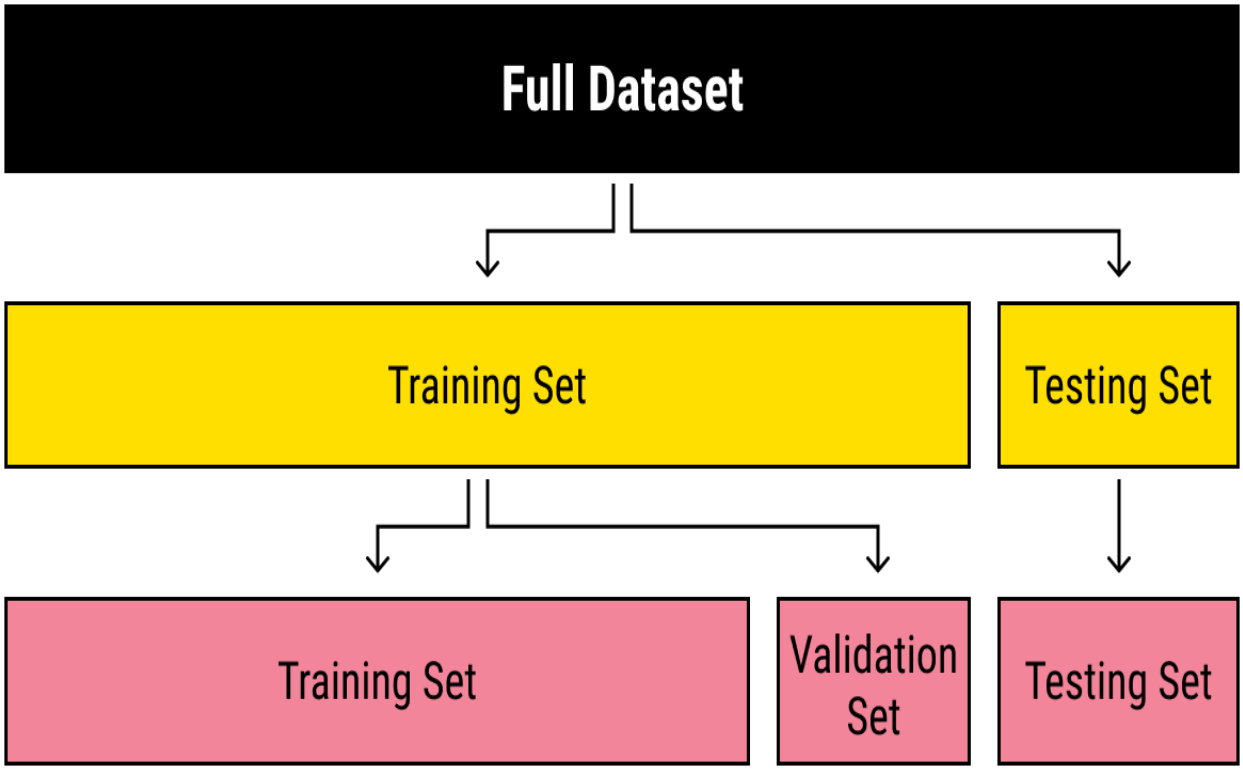
Data is an essential component of any AI model and, basically, the sole reason for the spike in popularity of machine learning that we witness today. Due to the availability of data, scalable ML algorithms became viable as actual products that can bring value to a business, rather than being a by-product of its main processes.

Your business has always been based on data. Factors such as what the customer bought, the popularity of the products, seasonality of the customer flow have always been important in business making. However, with the advent of machine learning, now it's important to collect this data into datasets.

Sufficient volumes of data allow you to analyze the trends and hidden patterns and make decisions based on the dataset you've built. However, while it may look rather simple, working with data is more complicated. It requires proper treatment of the data you have, from the purposes of using a dataset to the preparation of the raw data for it to be actually usable.

Splitting Your Data: Training, Testing, and Validation Datasets in Machine Learning

Usually, a dataset is used not only for training purposes. A single training set that has already been processed is usually split into several types of datasets in machine learning, which is needed to check how well the training of the model went. For this purpose, a testing dataset is typically separated from the data. Next, a validation dataset, while not strictly crucial, is quite helpful to avoid training your algorithm on the same type of data and making biased predictions.

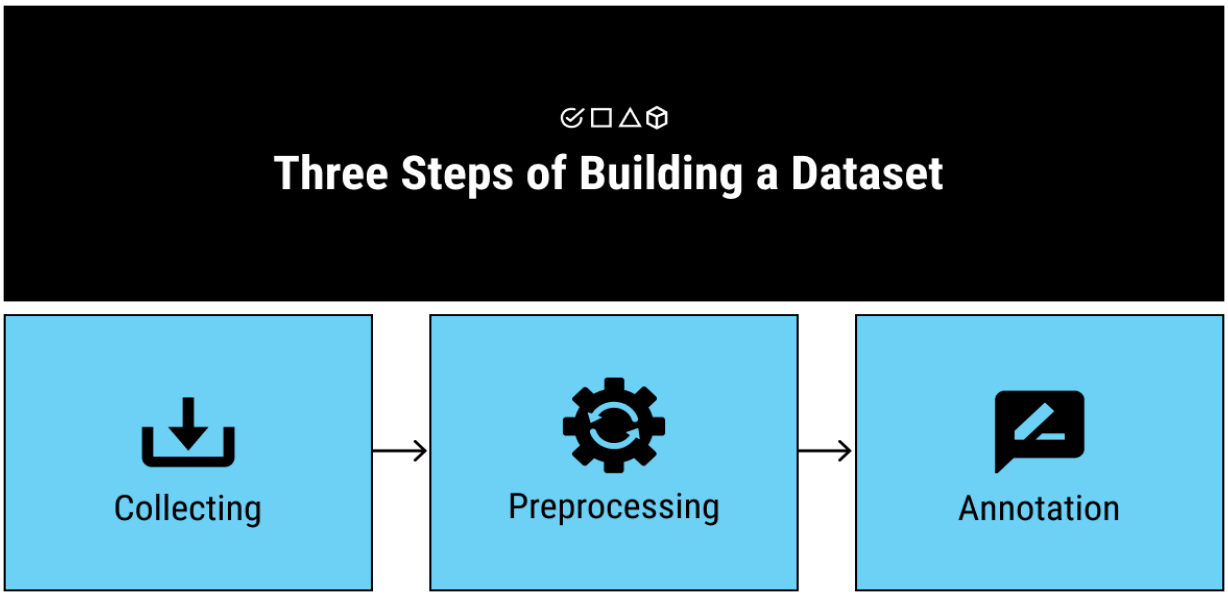


Splitting of a dataset into training, testing, and validation datasets

If you want to know more about how to split a dataset, we've covered this topic in detail in our article on training data.

Features of the Data: How to Build Yourself a Proper Dataset for a Machine Learning Project?

Raw data is a good place to start, but you obviously cannot just shove it into a machine learning algorithm and hope it offers you valuable insights into your customers' behaviors. There are quite a few steps you need to take before your dataset becomes usable.

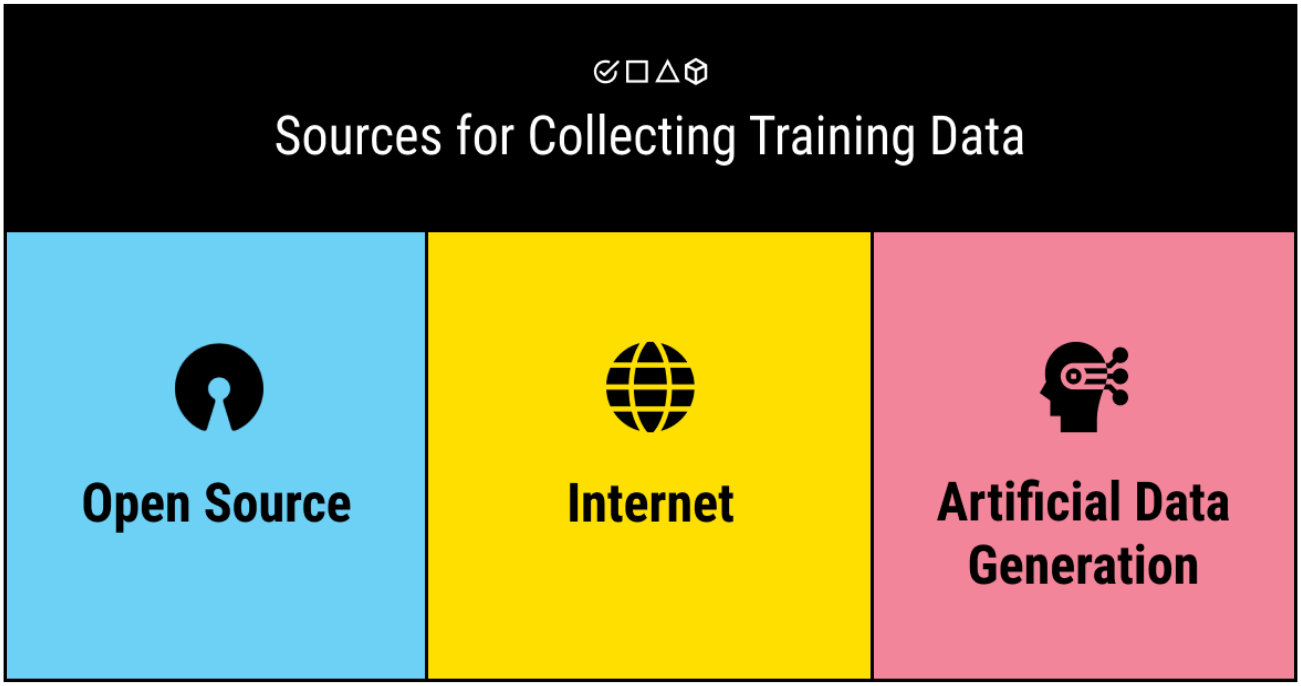


Three steps of data processing in machine learning

1. **Collect.** The first thing to do when you're looking for a dataset is deciding on the sources you'll be using for data collection in ML. Usually, there are three types of sources you can choose from: the freely available open-source datasets, the Internet, and the generators of artificial data. Each of these sources has its pros and cons and should be used for specific cases. We'll talk about this step in more detail in the next section of this article.
2. **Preprocess.** There's a principle in data science that every experienced professional adheres to. Start by answering this question: has the dataset you're using been used before? If not, assume this dataset is flawed. If yes, there's still a high probability you'll need to re-appropriate the set to fit your specific goals. After covering the sources, we'll talk more about the features that constitute a proper dataset (you can click here to skip to that section now).

3. **Annotate.** After you've ensured your data is clean and relevant, you also need to make sure it's understandable for a computer to process. Machines do not understand the data the same way as humans do (they aren't able to assign the same meaning to the images or words as we). This step is where a lot of businesses often decide to outsource the task to experienced data tagging services, since keeping a trained annotation professional is not always viable. We have a great article on building an in-house labeling team vs. outsourcing this task to help you understand which way is the best for you.

Quest for a Dataset in Machine Learning: Where to Find It and What Sources Fit Your Case Best?



The three sources of the dataset collection

The sources for collecting an AI/ML dataset vary and strongly depend on your project, budget, and size of your business. The best option is to get help from professional data collection services that directly correlate with your business goals. However, while this way you have the most control over the data that you collect, it may prove complicated and demanding in terms of financial, time, and human resources.

Other ways like automatically generated datasets require significant computational powers and are not suitable for any project. For the purposes of this article, we'd like to specifically distinguish the free, ready-to-use datasets for machine learning. There are large, comprehensive repositories of public datasets that can be freely downloaded and used for the training of your machine learning algorithm.

The obvious advantage of free datasets is that they're, well, free. On the other hand, you'll most likely need to tune any of such downloadable datasets to fit your project, since they were built for other purposes initially and won't fit precisely into your custom-built ML model. Still, this is an option of choice for many startups, as well as small and medium-sized businesses, since it requires fewer resources to collect a proper dataset.

The Features of a Proper, High-Quality Dataset in Machine Learning

A good dataset combines high quality with sufficient quantity

However, before you decide on what sources to use while collecting a dataset for your ML model, consider the following features of a good dataset.

Quality of a Dataset: Relevance and Coverage

High data quality is the essential thing to take into consideration when you collect a dataset for a machine learning project. But what does this mean in practice? First, the data pieces should be relevant to your goal. If you are designing an ML algorithm for an autonomous vehicle, you will have no need even for the best of datasets that consist of celebrity photos.

Furthermore, it's important to ensure the pieces of data are of sufficient quality. While there are ways of cleaning the data and making it uniform and manageable before annotation and training processes, it's best to have the data correspond to a list of required features. For example, when building a facial recognition model, you will need the training photos to be of good enough quality.

In addition, even for relevant and high-quality datasets, there is a problem of blind spots and biases that any data can be subject to. An imbalanced dataset in ML poses the dangers of throwing off the prediction results of your carefully built ML model. Let's say you're planning to build a text classification model to arrange a database of texts by topic. But if you only use NLP datasets that don't cover enough topics, your model will likely fail to recognize the rarer ones.

Tip: try to use live data and expert text annotation services. Fake data might seem like a good idea when you're building your model (it is cheaper, cleaner, and is available in large volumes). But if you try to cut costs by using a fake dataset, you might end up with a weirdly trained algorithm. Fake data might turn out to be too predictable or not predictable enough. Either way, it's not a great start for your AI project.

Sufficient Quantity of a Dataset in Machine Learning

Not only quality but quantity matters, too. It's important to have enough data to train your algorithm properly. There's also a possibility

of overtraining an algorithm (known as overfitting), but it's more likely you won't get enough high-quality data.

There's no perfect recipe for how much data you need. It's always a good idea to get advice from a data scientist. Professionals with extensive experience usually can roughly estimate the volume of the dataset you'll need for a specific AI project.

Alas, it is not sufficient to collect your dataset and make sure it corresponds to all the features we've listed above. There is one more step you need to take before starting the training of your ML model: analysis of the dataset.

There are cases that range from hilarious to horrifying about how strongly an ML algorithm depends on the exhaustive analysis of its dataset. One of such cases told by Martin Goodson, a guru of data science, shows the story of a hospital that decided to cut treatment costs for pneumonia patients. The highly accurate neural network that was built based on the clinic data could determine the patients with a low risk of developing complications. These patients could just take antibiotics at home without the need to visit the hospital.

However, when the model was considered for practical use, it was found that it sent all patients with asthma home even though these patients were actually at high risk of developing fatal complications.

The problem was that human doctors knew this and always sent such patients to intensive care. For this reason, the historic dataset of the hospital had no recorded deaths for asthmatics with pneumonia, which resulted in the algorithm deciding asthma was not an aggravating condition. If employed in a practical setting, the algorithm would potentially result in human deaths, even though the dataset was relevant, comprehensive, and of high quality.

This case demonstrates that machines still cannot do the analytic work of humans and are merely tools that require supervision and control. When your dataset is collected, cleansed, annotated, and seems ready, analyze it before deploying the data as a training tool for your model.

Collecting different types of datasets in machine learning might seem like an easy task that can be done in the background while you pour most of your time and resources into building the machine learning model. However, as practice shows, time and time again, dealing with data might take most of your time due to the sheer scale that this task might grow to. For this reason, it's important to understand what a dataset in machine learning is, how to collect the data, and what features a proper dataset has.

A machine learning dataset is, quite simply, a collection of data pieces that can be treated by a computer as a single unit for analytic and prediction purposes. This means that the data collected should be made uniform and understandable for a machine that doesn't see data the same way as humans do. For this, after collecting the data, it's

important to preprocess it by cleaning and completing it, as well as annotate the data by adding meaningful tags readable by a computer.

Moreover, a good dataset should correspond to certain quality and quantity standards. For smooth and fast training, you should make sure your dataset is relevant and well-balanced. Try to use live data whenever possible and consult with experienced professionals about the volume of the data and the source to collect it from.

Following these tips won't guarantee you collect a perfect dataset for your ML project. However, it will help you avoid some major pitfalls on your way to success.

Introduction to Proximity Measures

Proximity-Based Method

Proximity-based methods are an important technique in data mining. They are employed to find patterns in large databases by scanning documents for certain keywords and phrases. They are highly prevalent because they do not require expensive hardware or much storage space, and they scale up efficiently as the size of databases increases.

Advantages of Proximity-Based Methods:

1. Proximity-based methods make use of machine learning techniques, in which algorithms are trained to respond to certain patterns.
2. Using a random sample of documents, the machine learning algorithm analyzes the keywords and phrases used in them and makes predictions about the probability that these words appear together across all documents.
3. Proximity can be calculated by calculating a similarity score between two collections of training data and then comparing these scores. The algorithm then tries to compute the maximum similarity score for two distinct sets of training items.

Disadvantages of Proximity-Based Methods:

1. Important words may not be as close in proximity as we expected.
2. Over-segmentation of documents into phrases. To counter these problems, a lexical chain-based algorithm has been proposed.

Proximity-based methods perform very well for finding sets of documents that contain certain words based on background knowledge. But performance is limited when the background knowledge has not been pre-classified into categories.

To find sets of documents containing certain categories, one must assign categorical values to each document and then run proximity-based methods on these documents as training data, hoping for accurate representations of the categories.

One way to identify outliers is by calculating their distance from the rest of the data set in is known as density-based outlier detection.

Types of Proximity-Based Outlier Detection Methods:

- **Distance-based outlier detection methods:** A distance-based outlier detection method is a statistical technique. Such methods typically measure distances between individual data points and the rest of their respective groups. Many approaches also have a configurable error threshold for determining when a point is an outlier. Many distance-based outliers methods have been developed. The methods use distance statistics such as Euclidean, Manhattan, or Mahalanobis distance for calculating distances between individual points and to detect outliers. The following three outlier detection methods have been selected based on their performance:
    - WLSMV (Weighted Least Squares Minimization) method
    - SVM (Support Vector Machines) method,
    - RMSProp method.
- **Density-based Outlier detection methods:** A density-based outlier detection method is used for checking the density of an entity object and its closest objects. Key applications of this method are used in many applications including Malware Detection, Awareness, Behavior Analysis, and Network Intrusion Detection. There are some limitations to density-based outlier detection methods that are effective until it is determined that the outliers being detected are not necessarily outliers but just a part of a much larger distribution of data. A limitation with using density-based outlier

detection methods is that the density function must be defined and clearly understood before implementation and the proper value set.

Distance Measures

Measures of Distance

Clustering

consists of grouping certain objects that are similar to each other, it can be used to decide if two items are similar or dissimilar in their properties. In a Data Mining sense, the similarity measure is a distance with dimensions describing object features. That means if the distance among two data points is **small** then there is a **high** degree of similarity among the objects and vice versa. The similarity is **subjective** and depends heavily on the context and application. For example, similarity among vegetables can be determined from their taste, size, colour etc. Most clustering approaches use distance measures to assess the similarities or differences between a pair of objects, the most popular distance measures used are:

1. Euclidean Distance:

Euclidean distance is considered the traditional metric for problems with geometry. It can be simply explained as the **ordinary distance** between two points. It is one of the most used algorithms in the cluster analysis. One of the algorithms that use this formula would be **K-mean**. Mathematically it computes the **root of squared differences** between the coordinates between two objects.

d(p,q)=d(q,p)=(q1−p1)2+(q2−p2)2+⋯+(qn−pn)2=∑i=1n(qi−pi)2*d*(**p**,**q**)=*d*(**q**,**p**)=(*q*1−*p*1)2+(*q*2−*p*2)2+⋯+(*qn*−*pn*)2=*i*=1∑*n*(*qi*−*pi*)2

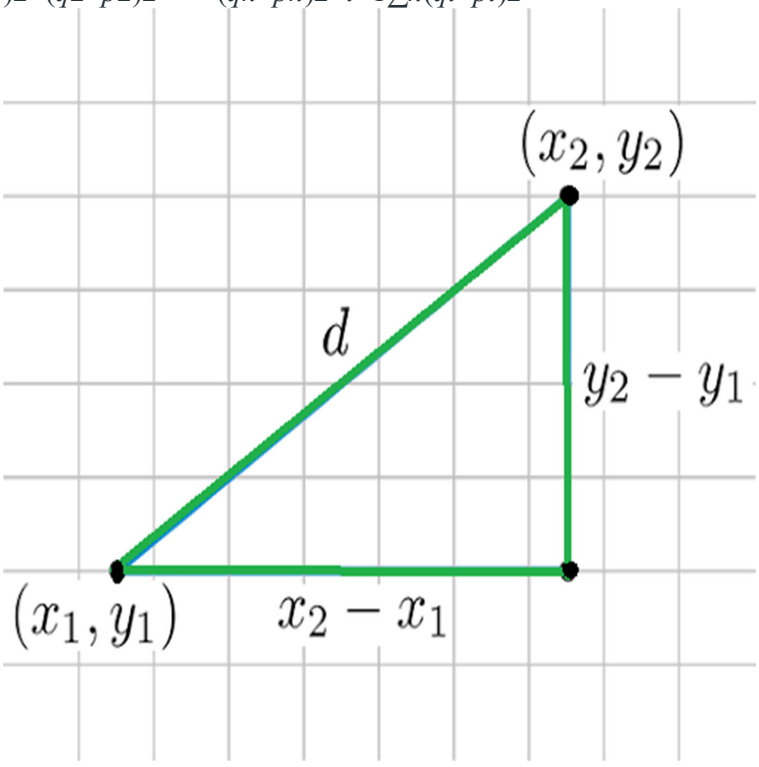


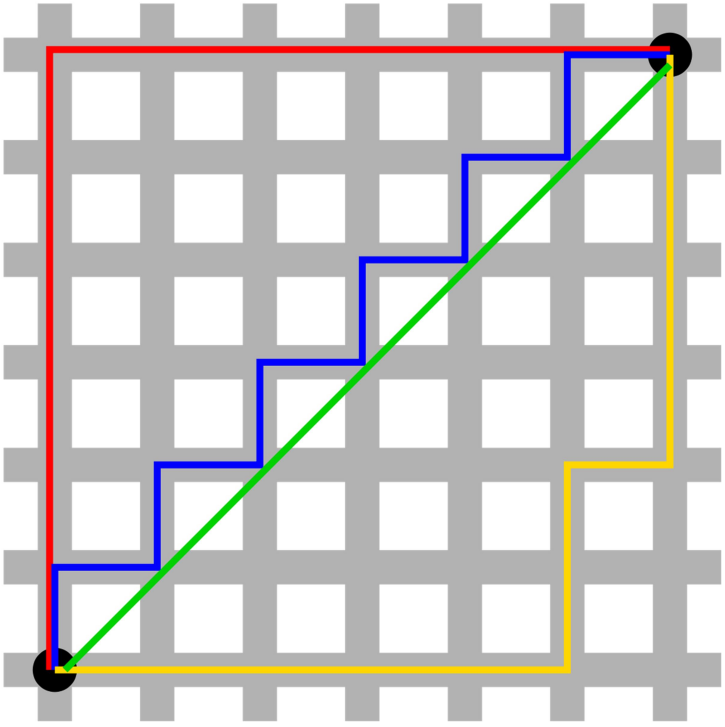
Figure –

Euclidean Distance

2. Manhattan Distance:

This determines the absolute difference among the pair of the coordinates. Suppose we have two points P and Q to determine the distance between these points we simply have to calculate the perpendicular distance of the points from X-Axis and Y-Axis. In a plane with P

at coordinate (x1, y1) and Q at (x2, y2). Manhattan distance between P and Q = |x1 – x2| + |y1 – y2|



Here the total distance of the **Red** line gives the Manhattan distance between both the points.

3. Jaccard Index:

The Jaccard distance measures the similarity of the two data set items as the **intersection** of those items divided by the **union** of the data items.

J(A,B)=|A∩B||A∪B|=|A∩B||A|+|B|−|A∩B|$J(A,B)=|A \cup B||A \cap B|=|A|+|B|-|A \cap B||A \cap B|$

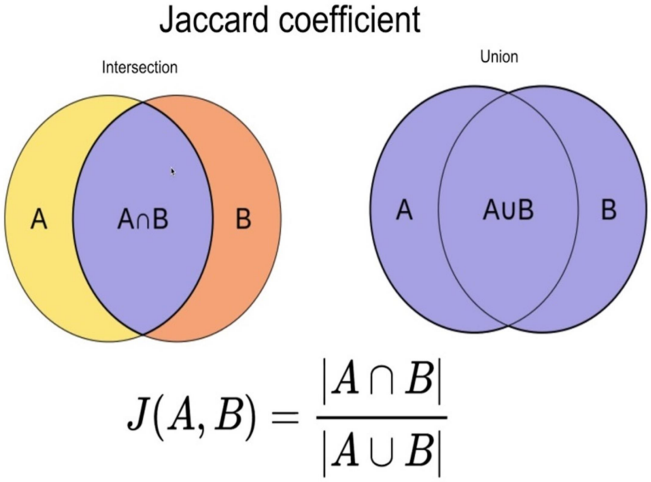


Figure –

Jaccard Index

4. Minkowski distance:

It is the **generalized** form of the Euclidean and Manhattan Distance Measure. In an **N-dimensional space**, a point is represented as,

(x1, x2, ..., xN)

Consider two points P1 and P2:

P1: (X1, X2, ..., XN)
P2: (Y1, Y2, ..., YN)

Then, the Minkowski distance between P1 and P2 is given as:

(x1−y1)p+(x2−y2)p+…+(xN−yN)pp$p(x1-y1)p+(x2-y2)p+\ldots+(xN-yN)p$

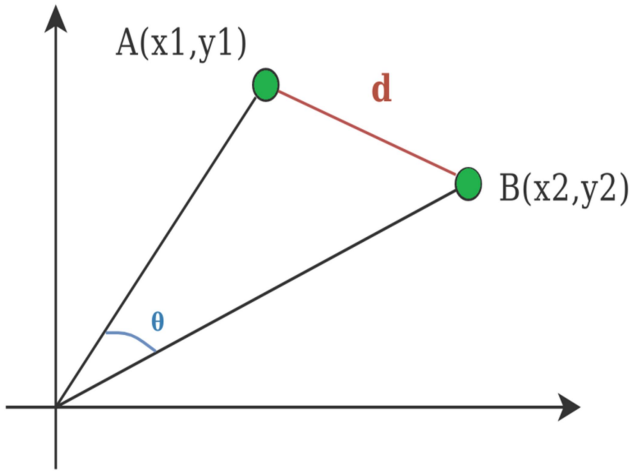
- When **p = 2**, Minkowski distance is same as the **Euclidean** distance.
- When **p = 1**, Minkowski distance is same as the **Manhattan** distance.

5. Cosine Index:

Cosine distance measure for clustering determines the **cosine** of the angle between two vectors given by the following formula.

sim (A,B)=cos (θ)=A·B‖A‖B‖sim$(A,B)=\cos(\theta)=\|A\|B\|A\cdot B$

Here (**theta**) gives the angle between two vectors and A, B are n-dimensional vectors.



Non-Metric Similarity Functions:

- **Definition:**
  These functions measure similarity or dissimilarity but do not necessarily satisfy all the axioms of a metric space, particularly the triangle inequality.
- **Examples:**
- **Cosine similarity:** Measures the cosine of the angle between two vectors, focusing on the direction rather than the magnitude.
- **Jaccard index:** Measures the similarity between two sets by calculating the ratio of the size of their intersection to the size of their union.
- **Squared Euclidean distance:** The square of the Euclidean distance, which is not a metric because it violates the triangle inequality.
- **Edit distance:** The minimum number of edits (insertions, deletions, or substitutions) required to transform one string into another.

Cosine Similarity

Cosine similarity is the measure of similarity between two non-zero vectors widely applied in many machine learning and data analysis applications. It actually measures the cosine of the angle between two vectors. As a result, an idea is given about how far the two vectors point in the same direction irrespective of their magnitudes. It can be found in popular usage in tasks of text analysis, such as comparison of similarity between documents, search queries, and even recommendation systems so that user preferences can be matched.

Similarity measure refers to distance with dimensions representing features of the data object, in a dataset. If this distance is less, there will be a high degree of similarity, but when the distance is large, there will be a low degree of similarity. Some of the popular similarity measures are given below:

1. Euclidean Distance
2. Manhattan Distance
3. Jaccard Similarity
4. Minkowski Distance
5. Cosine Similarity

What is Cosine Similarity?

Cosine similarity is a metric, helpful in determining, how similar the data objects are irrespective of their size. We can measure the similarity between two sentences in Python using Cosine Similarity. In cosine similarity, data objects in a dataset are treated as a vector. The formula to find the cosine similarity between two vectors is –

```
SCSc(x, y) = x . y / ||x|| ×× ||y||
```

where,

- **x . y** = product (dot) of the vectors 'x' and 'y'.
- ||**x**|| and ||**y**|| = length (magnitude) of the two vectors 'x' and 'y'.
- ||**x**|| ×× ||**y**|| = regular product of the two vectors 'x' and 'y'.

Example

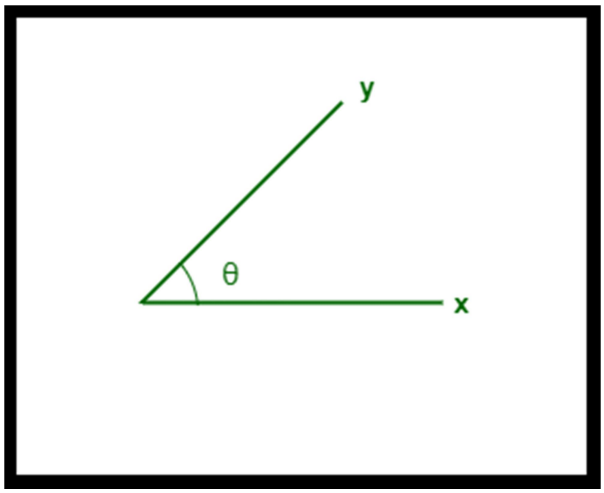
Consider an example to find the similarity between two vectors – **'x'** and **'y'**, using Cosine Similarity. The 'x' vector has values, **x = { 3, 2, 0, 5 }** The 'y' vector has values, **y = { 1, 0, 0, 0 }** The formula for calculating the cosine similarity is : SC*Sc* **(x, y) = x . y / ||x|| ×× ||y||**

```
x . y = 3*1 + 2*0 + 0*0 + 5*0 = 3

||x|| = √ (3)^2 + (2)^2 + (0)^2 + (5)^2 = 6.16

||y|| = √ (1)^2 + (0)^2 + (0)^2 + (0)^2 = 1

∴ SCSc(x, y) = 3 / (6.16 * 1) = 0.49
```

The dissimilarity between the two vectors 'x' and 'y' is given by –

```
∴ DCDc(x, y) = 1 - SCSc(x, y) = 1 - 0.49 = 0.51
```

- The cosine similarity between two vectors is measured in 'θ'.
- If θ = 0°, the 'x' and 'y' vectors overlap, thus proving they are similar.
- If θ = 90°, the 'x' and 'y' vectors are dissimilar.



Cosine Similarity between two vectors

Advantages

- The cosine similarity is beneficial because even if the two similar data objects are far apart by the Euclidean distance because of the size, they could still have a smaller angle between them. Smaller the angle, higher the similarity.
- When plotted on a multi-dimensional space, the cosine similarity captures the orientation (the angle) of the data objects and not the magnitude.

Disadvantages

- **Sensitive to Sparse Data:** Cosine similarity may not be effective when applied to the sparse data wherein many of its components are zero in the vectors. For that, other similarities would work better.
- **Does Not Account for Absolute Differences:** Cosine similarity only considers the angle between vectors and not their magnitude; hence, this may miss out on differences in magnitude, which, in certain contexts, may well be relevant
- **Symmetry:** Cosine similarity is symmetric, which simply means that it cannot differentiate between the order of comparison. For some tasks, this may not be desirable since directionality may be relevant.
- **Not Applicable for Negative Values:** Cosine similarity may not generally be applicable in datasets containing negative values, as misleading results can be obtained, or the angle between vectors interpretation becomes problematic.

Conclusion

Cosine similarity is also one of the approaches that are widely used as vector metrics in the field of text analysis and information retrieval. Cosine similarity has many advantages over the other measures of similarities – it is simple and efficient and can be treated with high-dimensional data.

How to Calculate Jaccard Similarity in Python

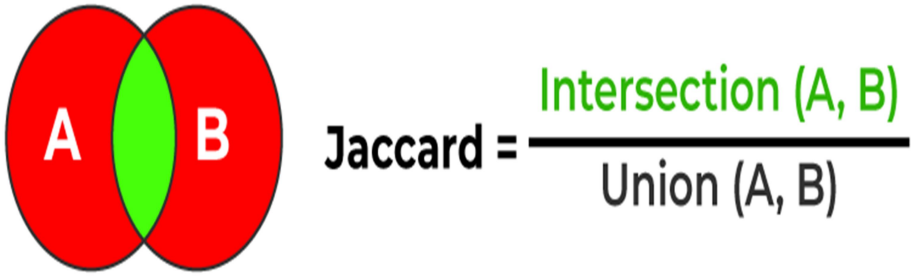
In Data Science, Similarity measurements between the two sets are a crucial task. Jaccard Similarity is one of the widely used techniques for similarity measurements in machine learning, natural language processing and

recommendation systems. This article explains what Jaccard similarity is, why it is important, and how to compute it with Python.

What is Jaccard Similarity?

Jaccard Similarity also known as Jaccard index, is a statistic to measure the similarity between two data sets. It is measured as the size of the intersection of two sets divided by the size of their union.

For example: Given two sets A and B, their Jaccard Similarity is provided by,



Jaccard Similarity

Where:

- is the cardinality (size) of the intersection of sets A and B.
- is the cardinality (size) of the union of sets A and B.

Jaccard Similarity is also known as the Jaccard index or Jaccard coefficient, its values lie between 0 and 1. where 0 means no similarity and the values get closer to 1 means increasing similarity 1 means the same datasets.

Euclidean Distance

Euclidean Distance is defined as the distance between two points in Euclidean space. To find the distance between two points, the length of the line segment that connects the two points should be measured.

In this article, we will explore what is Euclidean distance, the Euclidean distance formula, its Euclidean distance formula derivation, Euclidean distance examples, etc.

What is Euclidean Distance?

Euclidean distance is a measure of the straight-line distance between two points in Euclidean space. It is the most common and familiar distance metric, often referred to as the "ordinary" distance.

Euclidean Distance gives the distance between any two points in an n-dimensional plane. Euclidean distance between two points in the Euclidean space is defined as the length of the line segment joining the two points.

Euclidean distance is like **measuring the straightest and shortest path between two points**. Imagine you have a string and you stretch it tight between two points on a map; the length of that string is the Euclidean distance. It tells you how far

apart the two points are without any turns or bends, just like a bird would fly directly from one spot to another. This **metric is based on the Pythagorean theorem** and is widely utilized in various fields such as machine learning, data analysis, computer vision, and more.

Table of Content

Euclidean Distance Formula

Consider two points $(x_1, y1)$ and (x_2, y_2) in a 2-dimensional space; the Euclidean Distance between them is given by using the formula:

$$d = \sqrt{[(x_2 - x_1)_2 + (y_2 - y_1)_2]}$$

Where,

- d is Euclidean Distance
- (x_1, y_1) is Coordinate of the first point
- (x_2, y_2) is Coordinate of the second point

Euclidean Distance in 3D

If the two points (x_1, y_1, z_1) and (x_2, y_2, z_2) are in a 3-dimensional space, the Euclidean Distance between them is given by using the formula:

$$d = \sqrt{[(x_2 - x_1)_2 + (y_2 - y_1)_2 + (z_2 - z_1)_2]}$$

where,

- d is Euclidean Distance
- (x_1, y_1, z_1) is Coordinate of the first point
- (x_2, y_2, z_2) is Coordinate of the second point

Euclidean Distance in nD

In general, the Euclidean Distance formula between two points $(x_{11}, x_{12}, x_{13}, ..., x_{1n})$ and $(x_{21}, x_{22}, x_{23}, ..., x_{2n})$ in an n-dimensional space is given by the formula:

$$d = \sqrt{[\sum(x_{2i} - x_{1i})_2]}$$

Where,

- i Ranges from 1 to n
- d is Euclidean distance
- $(x_{11}, x_{12}, x_{13}, ..., x_{1n})$ is Coordinate of First Point
- $(x_{21}, x_{22}, x_{23}, ..., x_{2n})$ is Coordinate of Second Point

Euclidean Distance Formula Derivation

Euclidean Distance Formula is derived by following the steps added below:

Step 1: Let us consider two points, A (x_1, y_1) and B (x_2, y_2), and d is the distance between the two points.

Step 2: Join the points using a straight line (AB).
Step 3: Now, let us construct a right-angled triangle whose hypotenuse is AB, as shown in the figure below.

Step4: Now, using **Pythagoras theorem** we know that,

$$(Hypotenuse)^2 = (Base)^2 + (Perpendicular)^2$$

$$\Rightarrow d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

Now, take the square root on both sides of the equation, we get

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Proximity Between Binary Patterns

Proximity measures for binary attributes

Here's the sequence of steps to calculate proximity measures for binary attributes:

Step 1: Data representation

Suppose we have a table with the students' names corresponding to their end-semester results, showing whether they've passed or failed the specific courses. We want to see similarities or dissimilarities among students. Pass is represented by **P**, and the fail is represented by **F**.

Tabular Data

Student Name	English	Mathematics	Physics	Databases	Chemistry	Biology
John	P	P	F	P	F	P
David	P	P	P	F	F	P
Robert	F	P	F	P	P	F
Lisa	P	F	P	F	P	F
William	F	F	F	P	F	F

Step 2: Binary representation of data

Now, the next step is to convert the data into binary format. Since we have two attributes: pass and fail. Our example represents pass (P) as **1** and fail (F) as **0**. The updated table looks like this:

Binary Data

Student Name	English	Mathematics	Physics	Databases	Chemistry	Biology

John	1	1	0	1	0	1
David	1	1	1	0	0	1
Robert	0	1	0	1	1	0
Lisa	1	0	1	0	1	0
William	0	0	0	1	0	0

Step 3: Proximity measure selection

We first have to see if our data is symmetric: attributes that treat 0s and 1s equally, e.g., In our case, gender is a symmetric attribute because there's no inherent preference or value associated with one gender over the other; both male and female are treated equally in the dataset. Conversely, asymmetric attributes, where 0s and 1s hold different meanings, e.g., subjects and pass/fail outcomes, are asymmetric because 'fail' (0) often holds greater significance than 'pass' (1) in contexts like academic grading. We employ two distinct formulas for proximity measures for these attributes.

Symmetric attributes

For symmetric attributes, we have two objects (students in our case) and want to check the dissimilarity between their results. Let the two students be student $\mathrm{m}m$ and student $\mathrm{n}n$. We have the formula:
$\mathrm{d(m,n)=b+ca+b+c+e}.d(m,n)=a+b+c+eb+c.$

where
$\mathrm{a}\rightarrow\{\mathrm{m:1,n:1}\}.a\rightarrow\{m:1,n:1\}.$

The value of $\mathrm{a}a$ equals the number of all the courses the students $\mathrm{m}m$ and $\mathrm{n}n$ both have passed.
$\mathrm{b}\rightarrow\{\mathrm{m:1,n:0}\}b\rightarrow\{m:1,n:0\}$

The value of $\mathrm{b}b$ equals the number of all the courses where the student $\mathrm{m}m$ has passed and $\mathrm{n}n$ has failed.
$\mathrm{c}\rightarrow\{\mathrm{m:0,n:1}\}c\rightarrow\{m:0,n:1\}$

The value of $\mathrm{c}c$ equals the number of all the courses where the student $\mathrm{m}m$ has failed, and $\mathrm{n}n$ has passed.
$\mathrm{e}\rightarrow\{\mathrm{m:0,n:0}\}e\rightarrow\{m:0,n:0\}$

The value of $\mathrm{e}e$ equals the number of all the courses where the students $\mathrm{m}m$ and $\mathrm{n}n$ both have failed.

Asymmetric attributes

Suppose we have student m and student n for asymmetric attributes. Then the formula is:
$d(m,n)=\frac{b+c}{a+b+c}$.

Step 4: Dissimilarity calculation

As in our case, we only have asymmetric attributes, so we'll use that formula.

Let's calculate the dissimilarity for the pair, John and David.

- a **= 3** as both have passed English, Mathematics, and Biology courses.
- b **= 1** as John has passed the Databases course, and David has failed that.
- c **= 1** as John failed the Physics course, and David passed that.
- e **= 1** as both have failed in the Chemistry course.

So the dissimilarity is:
$d(m,n)=\frac{1+1}{3+1+1+1}=\frac{2}{5}=0.4$

Let's calculate the dissimilarity for the pair, Robert and William.

- a **= 1** as both have passed the Databases course.
- b **= 2** as Robert has passed the Chemistry and Mathematics courses, and William has failed those.
- c **= 0** (we have no such case here).
- e **= 3** as both have failed in the English, Physics, and Biology courses.

So the dissimilarity is:
$d(m,n)=\frac{2+0}{1+2+0+2+0}=\frac{2}{3}=0.667$

Similarly, after calculating the dissimilarity between the rest of the pairs, we get the following table:

Pair	Dissimilarity
John, David	0.4
John, Robert	0.6
John, Lisa	0.83
John, William	0.75
David, Robert	0.83
David, Lisa	0.6

David, William	1.0
Robert, Lisa	0.8
Robert, William	0.67
Lisa, William	1.0

Most dissimilar pairs (highest dissimilarity scores)

- David and William (dissimilarity score: 1.0)
- Lisa and William (dissimilarity score: 1.0)

Moderately dissimilar pairs

- John and Lisa (dissimilarity score: 0.83)
- David and Robert (dissimilarity score: 0.83)
- Robert and Lisa (dissimilarity score: 0.8)
- John and William (dissimilarity score: 0.75)

Moderately similar pairs

- David and Lisa (dissimilarity score: 0.6)
- Robert and William (dissimilarity score: 0.67)
- John and Robert (dissimilarity score: 0.6)

Most similar pairs (lowest dissimilarity score)

- John and David (dissimilarity score: 0.4)

Let’s quickly test your understanding of proximity measures for binary attributes.

Different Classification Algorithms Based on the Distance Measures

Several classification algorithms rely on distance measures to determine the similarity or dissimilarity between data points, aiding in the classification process. These include K-Nearest Neighbors (KNN), which classifies data based on the nearest neighbors, and Support Vector Machines (SVMs), which aim to maximize the distance between classes. Other algorithms like Naive Bayes also use distance metrics to find the closest feature.

Here's a more detailed look:

1. K-Nearest Neighbors (KNN):

- KNN is a simple algorithm that classifies new data based on its proximity to existing data points in a training set.
- It calculates the distance between a new data point and all existing data points and selects the k nearest neighbors.

- The new data point is then classified based on the majority class among its k nearest neighbors.
- Common distance metrics used in KNN include Euclidean distance, Manhattan distance, and Minkowski distance.

2. Support Vector Machines (SVMs):

- SVMs aim to find an optimal hyperplane that separates data points into different classes, maximizing the margin between them.
- The distance between the hyperplane and the nearest data points (support vectors) is crucial in SVM classification.
- SVMs can be used for both linear and non-linear classification problems.

3. Naive Bayes:

- Naive Bayes is a probabilistic classifier that uses Bayes' theorem to classify data.
- It relies on the assumption that the features are independent of each other.
- Distance metrics, like Euclidean distance or cosine similarity, can be used to find the closest feature when calculating probabilities.

4. Other Distance-Based Algorithms:

- **K-means clustering:**

  This algorithm is used for unsupervised learning, where it groups data points into clusters based on their proximity to cluster centroids.

- **Decision trees:**

  While not solely relying on distance, decision trees can use distance metrics to split data based on features, contributing to the classification process.

- **Random Forest:**
  This ensemble method combines multiple decision trees, and each tree can use distance metrics for splitting data, according to a resource on Towards Data Science.

Common Distance Metrics:

- **Euclidean distance:** Calculates the straight-line distance between two points.
- **Manhattan distance:** Calculates the distance between two points by summing the absolute differences of their coordinates.
- **Minkowski distance:** A generalization of Euclidean and Manhattan distances.
- **Cosine similarity:** Measures the similarity between two vectors by calculating the cosine of the angle between them.

- **Hamming distance:** Measures the dissimilarity between two binary strings by counting the number of positions where they differ.

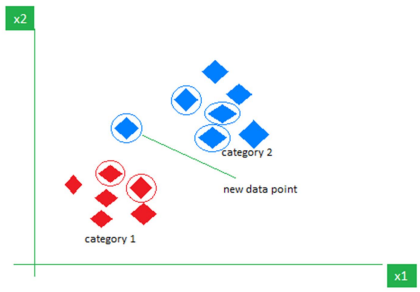
K-Nearest Neighbor(KNN)

K-Nearest Neighbors (KNN) is a simple way to classify things by looking at what's nearby. ***Imagine a streaming service wants to predict if a new user is likely to cancel their subscription (churn) based on their age***. They checks the ages of its existing users and whether they churned or stayed. ***If most of the "K" closest users in age of new user canceled their subscription KNN will predict the new user might churn too. The key idea is that users with similar ages tend to have similar behaviors and KNN uses this closeness to make decisions.***

Getting Started with K-Nearest Neighbors

K-Nearest Neighbors is also called as a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification it performs an action on the dataset.

As an example, consider the following table of data points containing two features:



KNN Algorithm working visualization

The new point is classified as **Category 2** because most of its closest neighbors are blue squares. KNN assigns the category based on the majority of nearby points. The image shows how KNN predicts the category of a **new data point** based on its closest neighbours.

- The **red diamonds** represent **Category 1** and the **blue squares** represent **Category 2**.
- The **new data point** checks its closest neighbours (circled points).
- Since the majority of its closest neighbours are blue squares (Category 2) KNN predicts the new data point belongs to Category 2.

KNN works by using proximity and majority voting to make predictions.

What is 'K' in K Nearest Neighbour ?

In the **k-Nearest Neighbours (k-NN)** algorithm **k** is just a number that tells the algorithm how many nearby points (neighbours) to look at when it makes a decision.

Example:

Imagine you're deciding which fruit it is based on its shape and size. You compare it to fruits you already know.

- If **k = 3**, the algorithm looks at the 3 closest fruits to the new one.
- If 2 of those 3 fruits are apples and 1 is a banana, the algorithm says the new fruit is an apple because most of its neighbours are apples.

How to choose the value of k for KNN Algorithm?

The value of k is critical in KNN as it determines the number of neighbors to consider when making predictions. Selecting the optimal value of k depends on the characteristics of the input data. **If the dataset has significant outliers or noise a higher k can help smooth out the predictions and reduce the influence of noisy data. However choosing very high value can lead to underfitting where the model becomes too simplistic.**

Statistical Methods for Selecting k:

- **Cross-Validation**: A robust method for selecting the best k is to perform k-fold cross-validation. This involves splitting the data into k subsets training the model on some subsets and testing it on the remaining ones and repeating this for each subset. The value of k that results in the highest average validation accuracy is usually the best choice.
- **Elbow Method**: In the **elbow method** we plot the model's error rate or accuracy for different values of k. As we increase k the error usually decreases initially. However after a certain point the error rate starts to decrease more slowly. This point where the curve forms an "elbow" that point is considered as best k.
- **Odd Values for k**: It's also recommended to choose an odd value for k especially in classification tasks to avoid ties when deciding the majority class.

Distance Metrics Used in KNN Algorithm

KNN uses distance metrics to identify nearest neighbour, these neighbours are used for classification and regression task. To identify nearest neighbour we use below distance metrics:

1. Euclidean Distance

Euclidean distance is defined as the straight-line distance between two points in a plane or space. You can think of it like the shortest path you would walk if you were to go directly from one point to another.

distance(x,Xi)=∑j=1d(xj–Xij)2]distance(x,X_i)=$\sum_{j=1}d(x_j–X_{ij})2$]

2. Manhattan Distance

This is the total distance you would travel if you could only move along horizontal and vertical lines (like a grid or city streets). It's also called "taxicab distance" because a taxi can only drive along the grid-like streets of a city.

d(x,y)=∑i=1n|xi−yi|$d(x,y)=\sum_{i=1n}|x_i−y_i|$

3. Minkowski Distance

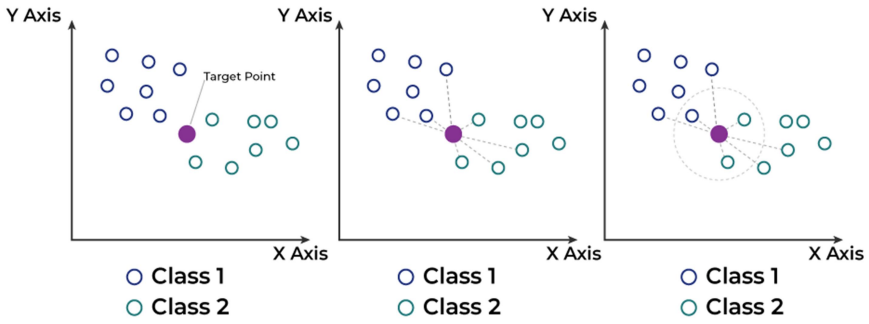
Minkowski distance is like a family of distances, which includes both **Euclidean** and **Manhattan distances** as special cases.

d(x,y)=(∑i=1n(xi−yi)p)1p$d(x,y)=(\sum_{i=1n}(x_i−y_i)_p)_{p1}$

From the formula above we can say that when p = 2 then it is the same as the formula for the Euclidean distance and when p = 1 then we obtain the formula for the Manhattan distance.
So, you can think of Minkowski as a flexible distance formula that can look like either Manhattan or Euclidean distance depending on the value of p

Working of KNN algorithm

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Step-by-Step explanation of how KNN works is discussed below:

Step 1: Selecting the optimal value of K

- K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

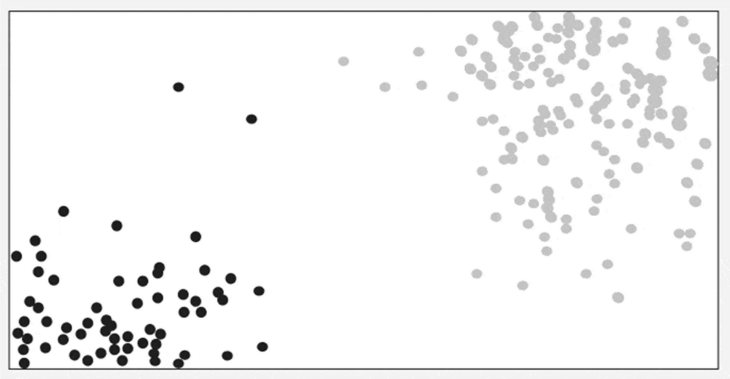
- To measure the similarity between target and training data points Euclidean distance is used. Distance is calculated between data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

- The k data points with the smallest distances to the target point are nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

- When you want to classify a data point into a category (like spam or not spam), the K-NN algorithm looks at the **K closest points** in the dataset. These closest points are called neighbors. The algorithm then looks at which category the neighbors belong to and picks the one that appears the most. This is called **majority voting**.
- In regression, the algorithm still looks for the **K closest points**. But instead of voting for a class in classification, it takes the **average** of the values of those K neighbors. This average is the predicted value for the new point for the algorithm.



Working of KNN Algorithm

It shows how a test point is classified based on its nearest neighbors. As the test point moves the algorithm identifies the closest 'k' data points i.e 5 in this case and assigns test point the majority class label that is grey label class here.

Applications of the KNN Algorithm

Here are some real life applications of KNN Algorithm.

- **Recommendation Systems**: Many recommendation systems, such as those used by Netflix or Amazon, rely on KNN to suggest products or content. KNN observes at user behavior and finds similar users. If user A and user B have similar preferences, KNN might recommend movies that user A liked to user B.
- **Spam Detection**: KNN is widely used in filtering spam emails. By comparing the features of a new email with those of previously labeled spam and non-spam emails, KNN can predict whether a new email is spam or not.
- **Customer Segmentation**: In marketing firms, KNN is used to segment customers based on their purchasing behavior . By comparing new customers to existing customers, KNN can easily group customers into segments with similar choices and preferences. This helps businesses target the right customers with right products or advertisements.
- **Speech Recognition**: KNN is often used in speech recognition systems to transcribe spoken words into text. The algorithm compares the features of the spoken input with those of known speech patterns. It then predicts the most likely word or command based on the closest matches.

Advantages and Disadvantages of the KNN Algorithm

Advantages:

- **Easy to implement:** The KNN algorithm is easy to implement because its complexity is relatively low as compared to other machine learning algorithms.
- **No training required:** KNN stores all data in memory and doesn't require any training so when new data points are added it automatically adjusts and uses the new data for future predictions.
- **Few Hyperparameters:** The only parameters which are required in the training of a KNN algorithm are the value of k and the choice of the distance metric which we would like to choose from our evaluation metric.

- **Flexible**: It works for **Classification** problem like is this email spam or not? and also work for **Regression task** like predicting house prices based on nearby similar houses.

Disadvantages:

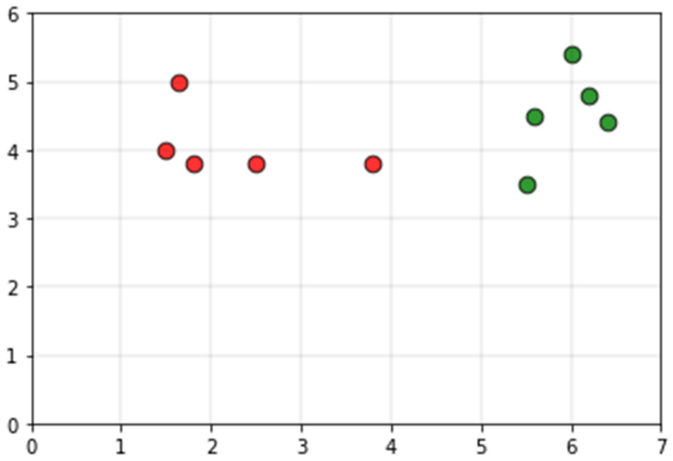
- **Doesn't scale well:** KNN is considered as a "lazy" algorithm as it is very slow especially with large datasets
- **Curse of Dimensionality:** When the number of features increases KNN struggles to classify data accurately a problem known as curse of dimensionality.
- **Prone to Overfitting:** As the algorithm is affected due to the curse of dimensionality it is prone to the problem of overfitting as well

r-Nearest neighbors

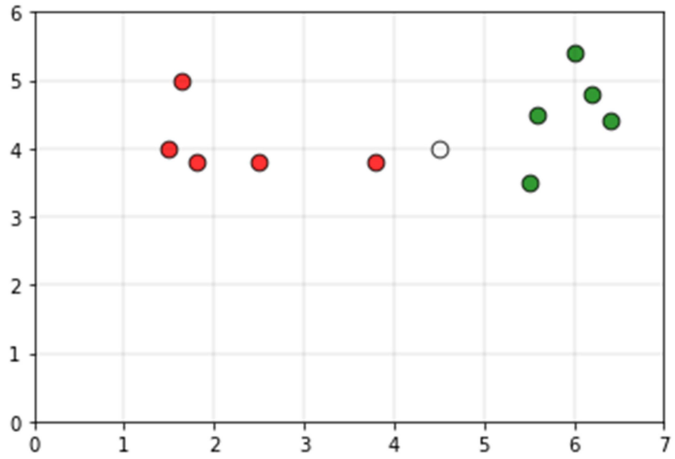
r-Nearest neighbors are a modified version of the k-nearest neighbors. The issue with k-nearest neighbors is the choice of k. With a smaller k, the classifier would be more sensitive to outliers. If the value of k is large, then the classifier would be including many points from other classes. It is from this logic that we get the r near neighbors algorithm.

Intuition:

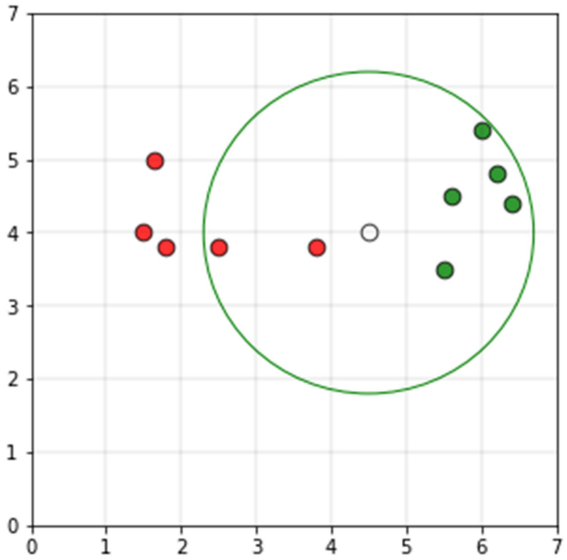
Consider the following data, as the training set.



The green color points belong to class 0 and the red color points belong to class 1. Consider the white point P as the query point whose



If we take the radius of the circle as 2.2 units and if a circle is drawn using the point P as the center of the circle, the plot would be as follows



As the number of points in the circle belonging to class 1 (5 points) is greater than the number of points belonging to class 0 (2 points)

Algorithm:

Step 1: *Given the point P, determine the sub-set of data that lies in the ball of radius r centered at P,*

$$B_r(P) = \{ X_i \in X \mid dist(P, X_i) \leq r \}$$

Step 2: *If $B_r(P)$ is empty, then output the majority class of the entire data set.*

Step 3: *If $B_r(P)$ is not empty, output the majority class of the data points in it.*

K-Nearest Neighbors (KNN) Regression with Scikit-Learn

K-Nearest Neighbors (KNN) is one of the simplest and most intuitive machine learning algorithms. While it is commonly associated with classification tasks, KNN can also be used for regression.

This article will delve into the fundamentals of KNN regression, how it works, and how to implement it using Scikit-Learn, a popular machine learning library in Python.

What is KNN Regression?

KNN regression is a non-parametric method used for predicting continuous values. The core idea is to predict the target value for a new data point by averaging the target values of the K nearest neighbors in the feature space. The distance between data points is typically measured using Euclidean distance, although other distance metrics can be used.

How KNN Regression Works

1. **Choosing the number of neighbors (K)**: The initial step involves selecting the number of neighbors, K. This choice greatly affects the model's performance. A smaller value of K makes the model more prone to noise, whereas a larger value of K results in smoother predictions.
2. **Calculating distances**: For a new data point, calculate the distance between this point and all points in the training set.
3. **Finding K nearest neighbors**: Identify the K points in the training set that are closest to the new data point.
4. **Predicting the target value**: Compute the average of the target values of the K nearest neighbors and use this as the predicted value for the new data point.

Implementing KNN Regression with Scikit-Learn using Synthetic Dataset

Let's go through a practical example of implementing KNN regression using Scikit-Learn. We will use a synthetic dataset for demonstration purposes.

Step 1: Import Libraries

In this step, we import the necessary libraries for generating the dataset, splitting the data, training the KNN model, evaluating the model, and visualizing the results.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

Step 2: Generate Synthetic Dataset

Here, we generate a synthetic dataset using Scikit-Learn's `make_regression` function. This function creates a regression problem with a specified number of samples, features, and noise level.

```
# Generate synthetic dataset
X, y = make_regression(n_samples=200, n_features=1, noise=0.1,
random_state=42)
```

Step 3: Split the Dataset

We split the dataset into training and testing sets using the `train_test_split` function. This step ensures that we have separate data for training the model and evaluating its performance.

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Step 4: Create and Train the KNN Regressor

In this step, we create an instance of the `KNeighborsRegressor` with a specified number of neighbors (K=5). We then train the model using the training data.

```
# Create and train the KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(X_train, y_train)
```

Step 5: Make Predictions

We use the trained KNN regressor to make predictions on the test data.

```
# Make predictions on the test data
y_pred = knn_regressor.predict(X_test)
```

Step 6: Evaluate the Model

Here, we evaluate the model's performance using the Mean Squared Error (MSE) and R-squared (R^2) metrics. These metrics help us understand how well the model is performing.

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Step 7: Visualize the Results

Finally, we visualize the actual and predicted values using a scatter plot. This step helps us visually assess the model's performance.

```
# Visualize the results
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.scatter(X_test, y_pred, color='red', label='Predicted')
plt.title('KNN Regression')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.legend()
plt.show()
```

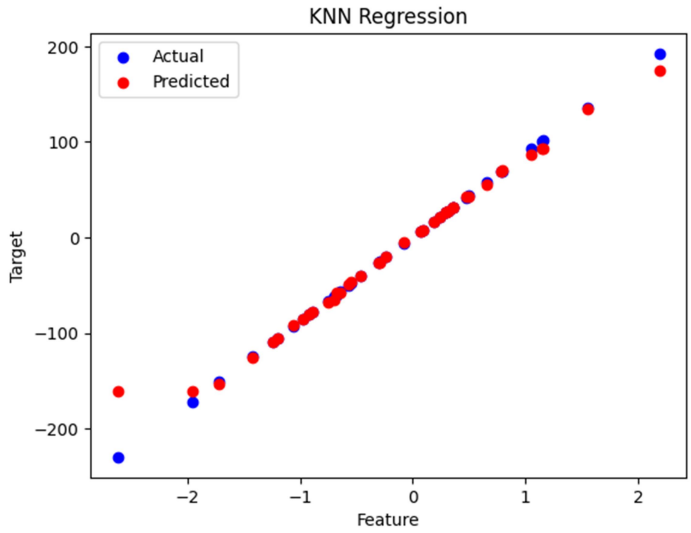
Complete Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Generate synthetic dataset
X, y = make_regression(n_samples=200, n_features=1, noise=0.1,
random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and train the KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn_regressor.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Visualize the results
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.scatter(X_test, y_pred, color='red', label='Predicted')
plt.title('KNN Regression')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.legend()
plt.show()
```

Output:

```
Mean Squared Error: 133.62045142000457
R-squared: 0.9817384115764595
```



KNN Regression

Implementing KNN Regression with Scikit-Learn using Diabetes Dataset

Let's use the diabetes dataset to perform KNN regression using the following steps:

Step 1: Import Libraries

In this step, we import the necessary libraries for loading the dataset, splitting the data, training the KNN model, evaluating the model, and visualizing the results.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
```

Step 2: Load the Dataset

Here, we load the Diabetes dataset using Scikit-Learn's `load_diabetes` function. This dataset includes ten baseline variables and a target variable representing the disease progression.

```
# Load the Diabetes dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# Print dataset description
print(diabetes.DESCR)
```

Step 3: Split the Dataset

We split the dataset into training and testing sets using the `train_test_split` function. This step ensures that we have separate data for training the model and evaluating its performance.

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Step 4: Standardize the Features

In this step, we standardize the features using `StandardScaler`. Standardization ensures that each feature has a mean of 0 and a standard deviation of 1, which helps improve the performance of the KNN algorithm.

```
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Step 5: Create and Train the KNN Regressor

We create an instance of the `KNeighborsRegressor` with a specified number of neighbors (K=5) and train the model using the training data.

```
# Create and train the KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(X_train, y_train)
```

Step 6: Make Predictions

We use the trained KNN regressor to make predictions on the test data.

```
# Make predictions on the test data
y_pred = knn_regressor.predict(X_test)
```

Step 7: Evaluate the Model

Here, we evaluate the model's performance using the Mean Squared Error (MSE) and R-squared (R^2) metrics. These metrics help us understand how well the model is performing.

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Step 8: Visualize the Results

Finally, we visualize the actual and predicted values using a scatter plot. This step helps us visually assess the model's performance.

```
# Visualize the results
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', label='Predicted vs
Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', linewidth=2, label='Ideal fit')
plt.title('KNN Regression: Predicted vs Actual')
plt.xlabel('Actual Disease Progression')
```

```
plt.ylabel('Predicted Disease Progression')
plt.legend()
plt.show()
```

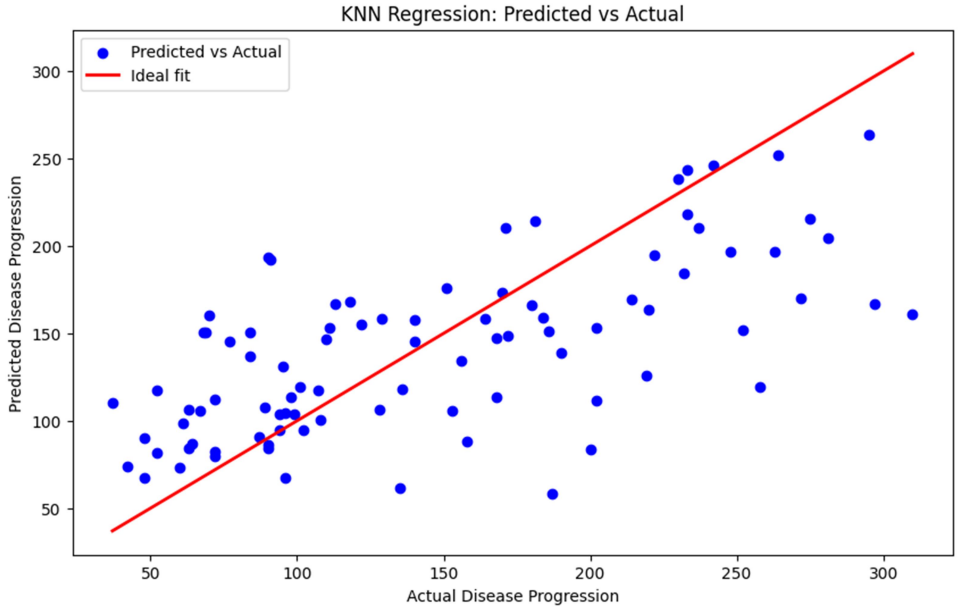
Complete Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Load the Diabetes dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# Print dataset description
print(diabetes.DESCR)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn_regressor.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Visualize the results
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
linewidth=2, label='Ideal fit')
plt.title('KNN Regression: Predicted vs Actual')
plt.xlabel('Actual Disease Progression')
plt.ylabel('Predicted Disease Progression')
```

```
plt.legend()
plt.show()
```

Output:

```
Mean Squared Error: 3047.449887640449
R-squared: 0.42480887066066253
```



Regression in machine learning

Regression in machine learning refers to a supervised learning technique where the goal is to predict a continuous numerical value based on one or more independent features. It finds relationships between variables so that predictions can be made. we have two types of variables present in regression:

- Dependent Variable (Target): The variable we are trying to predict e.g house price.
- Independent Variables (Features): The input variables that influence the prediction e.g locality, number of rooms.

Regression analysis problem works with if output variable is a real or continuous value such as “salary” or “weight”. Many different regression models can be used but the simplest model in them is linear regression.

Types of Regression

Regression can be classified into different types based on the number of predictor variables and the nature of the relationship between variables:

1. Simple Linear Regression

Linear regression is one of the simplest and most widely used statistical models. This assumes that there is a linear relationship between the independent and dependent variables. This means that the change in the dependent variable is proportional to the change in the independent variables. For example predicting the price of a house based on its size.

2. Multiple Linear Regression

Multiple linear regression extends simple linear regression by using multiple independent variables to predict target variable. For example predicting the price of a house based on multiple features such as size, location, number of rooms, etc.

3. Polynomial Regression

Polynomial regression is used to model with non-linear relationships between the dependent variable and the independent variables. It adds polynomial terms to the linear regression model to capture more complex relationships. For example when we want to predict a non-linear trend like population growth over time we use polynomial regression.

4. Ridge & Lasso Regression

Ridge & lasso regression are regularized versions of linear regression that help avoid overfitting by penalizing large coefficients. When there's a risk of overfitting due to too many features we use these type of regression algorithms.

5. Support Vector Regression (SVR)

SVR is a type of regression algorithm that is based on the Support Vector Machine (SVM) algorithm. SVM is a type of algorithm that is used for classification tasks but it can also be used for regression tasks. SVR works by finding a hyperplane that minimizes the sum of the squared residuals between the predicted and actual values.

6. Decision Tree Regression

Decision tree Uses a tree-like structure to make decisions where each branch of tree represents a decision and leaves represent outcomes. For example predicting customer behavior based on features like age, income, etc there we use decison tree regression.

7. Random Forest Regression

Random Forest is a ensemble method that builds multiple decision trees and each tree is trained on a different subset of the training data. The final prediction is made by averaging the predictions of all of the trees. For example customer churn or sales data using this.

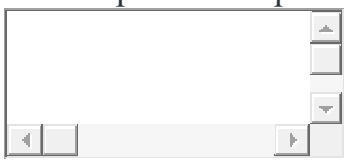
Regression Evaluation Metrics

Evaluation in machine learning measures the performance of a model. Here are some popular evaluation metrics for regression:

- Mean Absolute Error (MAE): The average absolute difference between the predicted and actual values of the target variable.
- Mean Squared Error (MSE): The average squared difference between the predicted and actual values of the target variable.
- Root Mean Squared Error (RMSE): Square root of the mean squared error.
- Huber Loss: A hybrid loss function that transitions from MAE to MSE for larger errors, providing balance between robustness and MSE's sensitivity to outliers.
- R2 – Score: Higher values indicate better fit ranging from 0 to 1.

Regression Model Machine Learning

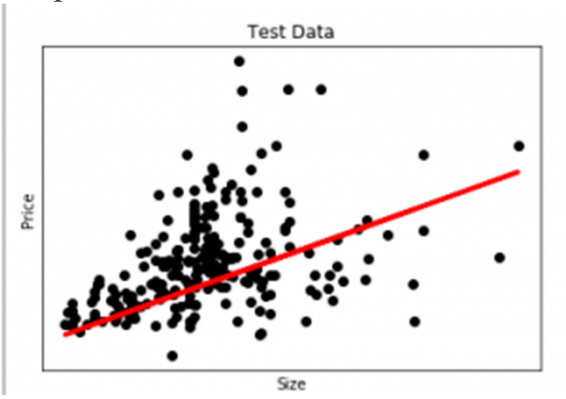
Let's take an example of linear regression. We have a Housing data set and we want to predict the price of the house. Following is the python code for it



```
import matplotlib
matplotlib.use('TkAgg')  # General backend for plots
 import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
import pandas as pd
 # Load dataset
df = pd.read_csv("Housing.csv")
# Extract features and target variable
Y = df['price']
X = df['lotsize']
# Reshape for compatibility with scikit-learn
X = X.to_numpy().reshape(len(X), 1)
Y = Y.to_numpy().reshape(len(Y), 1)
# Split data into training and testing sets
X_train = X[:-250]
X_test = X[-250:]
```

```
Y_train = Y[:-250]
Y_test = Y[-250:]


# Plot the test data
plt.scatter(X_test, Y_test, color='black')
plt.title('Test Data')
plt.xlabel('Size')
plt.ylabel('Price')
plt.xticks(())
plt.yticks(())
# Train linear regression model
regr = linear_model.LinearRegression()
regr.fit(X_train, Y_train)
# Plot predictions
plt.plot(X_test, regr.predict(X_test), color='red', linewidth=3)
plt.show()
```

Output:



Here in this graph we plot the test data. The red line indicates the best fit line for predicting the price.

To make an individual prediction using the linear regression model:

```
print("Predicted price for a lot size of 5000: " +
str(round(regr.predict([[5000]])[0][0])))
```

Applications of Regression

- Predicting prices: Used to predict the price of a house based on its size, location and other features.
- Forecasting trends: Model to forecast the sales of a product based on historical sales data.
- Identifying risk factors: Used to identify risk factors for heart patient based on patient medical data.
- Making decisions: It could be used to recommend which stock to buy based on market data.

Advantages of Regression

- Easy to understand and interpret.
- Robust to outliers.
- Can handle both linear relationships easily.

Disadvantages of Regression

- Assumes linearity.
- Sensitive to situation where two or more independent variables are highly correlated with each other i.e multicollinearity.
- May not be suitable for highly complex relationships.

Conclusion

Regression in machine learning is a fundamental technique for predicting continuous outcomes based on input features. It is used in many real-world applications like price prediction, trend analysis and risk assessment. With its simplicity and effectiveness regression is used to understand relationships in data.

Decision Trees for Classification

Decision trees are a popular supervised machine learning algorithm used for both classification and regression, where classification trees predict categorical outcomes by following a tree-like structure of decisions based on data features. Here's a more detailed explanation:

Key Concepts:

Tree Structure:

Decision trees are visualized as a tree, with:

- **Root Node:** The starting point of the tree.
- **Internal Nodes:** Represent features or attributes used for making decisions.
- **Branches:** Represent possible outcomes or values of the feature.
- **Leaf Nodes:** Represent the final classification or prediction.

Classification:

In classification, each leaf node represents a class label, and the algorithm classifies an instance by following the branches from the root to a leaf node.

Supervised Learning:

Decision trees are a type of supervised learning algorithm, meaning they learn from labeled data to make predictions.

Recursive Partitioning:

Decision trees work by recursively partitioning the data into subsets based on feature values, creating a tree structure that represents the decision rules.

Advantages:

- **Interpretability:** Decision trees are relatively easy to understand and interpret, making them suitable for explaining predictions.
- **Handles both numerical and categorical data:** Decision trees can handle both types of data without requiring much preprocessing.
- **Can handle high-dimensional data:** Decision trees can handle a large number of features with good accuracy.

Disadvantages:

- **Overfitting:** Decision trees can be prone to overfitting, meaning they learn the training data too well and perform poorly on new, unseen data.
- **Sensitivity to small variations in data:** Small changes in the training data can lead to significant changes in the tree structure.

Ensemble Methods:

To address the limitations of individual decision trees, ensemble methods like Random Forests and Gradient Boosting are often used, which combine multiple trees to improve accuracy and robustness.

Scikit-learn:

The scikit-learn library provides a powerful implementation of decision tree algorithms, including the DecisionTreeClassifier class for classification tasks.

1.10.1. Classification

DecisionTreeClassifier is a class capable of performing multi-class classification on a dataset.

As with other classifiers, **DecisionTreeClassifier** takes as input two arrays: an array X, sparse or dense, of shape (n_samples, n_features) holding the training samples, and an array Y of integer values, shape (n_samples,), holding the class labels for the training samples:

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```

After being fitted, the model can then be used to predict the class of samples:

```
>>> clf.predict([[2., 2.]])
array([1])
```

In case that there are multiple classes with the same and highest probability, the classifier will predict the class with the lowest index amongst those classes.

As an alternative to outputting a specific class, the probability of each class can be predicted, which is the fraction of training samples of the class in a leaf:

```
>>> clf.predict_proba([[2., 2.]])
array([[0., 1.]])
```

DecisionTreeClassifier is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification.

Using the Iris dataset, we can construct a tree as follows:

```
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, y)
```

Once trained, you can plot the tree with the **plot_tree** function:

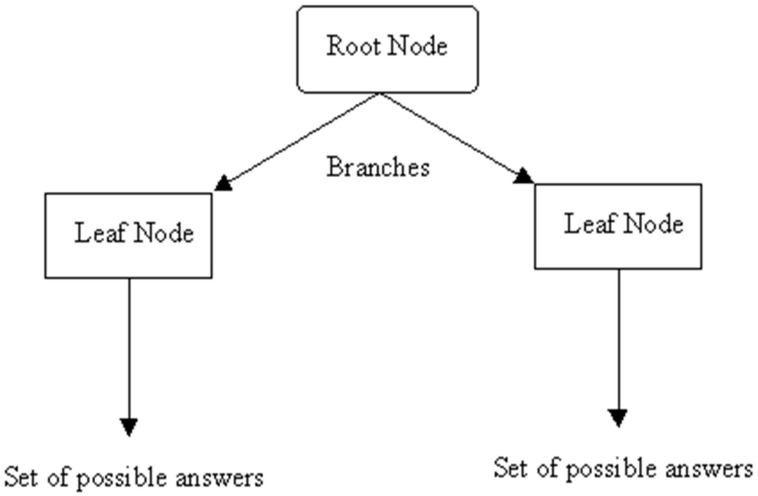
```
>>> tree.plot_tree(clf)
[...]
```

Impurity Measures

Impurity measures are used in Decision Trees just like squared loss function in linear regression. We try to arrive at as lowest impurity as possible by the algorithm of our choice. *Impurity is presence of more than one class in a subset of data.*

So all below mentioned measures differ in formula but align in goal. Watch till the end to know secret highlights of this topic.

Remember this



Make sure you understand that impurity measure is calculated for each leaf node, and its weighted average is the corresponding impurity measure for root node, based on which we say that this feature would become decision feature or not.

Let's take an example with Entropy and solve to see the exact formulation.

Entropy

The formula for impurity at leaf node is

$$E(S) = \sum_{1}^{c} -p_i \log_2 p_i$$

After taking weighted average for a feature, we need to check if this feature brings the most reduction in impurity. While using Entropy we do this by *Information Gain*

$$Information\ Gain = E(Y) - E(Y|X))$$

where

E(Y) should be Entropy before splitting the data over X

E(Y|X) is Weighted Entropy after split over X

Example: Consider the Contingency Table asdvv

Credit Rating	Liability		
	Normal	High	Total
Excellent	3	1	4
Good	4	2	6
Poor	0	4	4
Total	7	7	14

This should be read as simply Horizontal division is of Liability class labels(Normal and High), while vertical division is that of Credit Rating(Excellent, Good, Poor). So the number 3 in table implies, that out of total 14 companies there were 3 companies which got 'Excellent rating' and had 'Normal Liability'.
Calculation of Entropy for deciding if Credit Rating should be the first split.
First calculate Entropy before splitting

$$E(Liability) = -\frac{7}{14}\log_2\left(\frac{7}{14}\right) - \frac{7}{14}\log_2\left(\frac{7}{14}\right)$$

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

$$= 1$$

Next entropy over each Leaf node and then weighted average over credit rating split

$$E(Liability \mid CR = Excellent) = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) \approx 0.811$$

$$E(Liability \mid CR = Good) = -\frac{4}{6}\log_2\left(\frac{4}{6}\right) - \frac{2}{6}\log_2\left(\frac{2}{6}\right) \approx 0.918$$

$$E(Liability \mid CR = Poor) = -0\log_2(0) - \frac{4}{4}\log_2\left(\frac{4}{4}\right) = 0$$

Weighted Average:

$$E(Liability \mid CR) = \frac{4}{14} \times 0.811 + \frac{6}{14} \times 0.918 + \frac{4}{14} \times 0$$

$$= 0.625$$

Note greater the entropy, worse is the current feature for split at present level.
We now calculate information gain(Higher the better, or lower the conditional entropy)

Information Gain:

$$IG(Liability, CR) = E(Liability) - E(Liability \mid CR)$$

$$= 1 - 0.625$$

$$= 0.375$$

So we get 0.375 as the IG from Credit Rating as the metric for classification of data over liability status. If we had suppose stock price as a independent feature, we would have done the same thing for it as well. Then we would have compared the result for both, and one with higher information gain would have been our first decision variable for splitting.
Now
Impurity Reduction = G(Y) — G(Y|X))
Gini Index
The formula for leaf node is

$$G(S) = 1 - \sum_{1}^{c} p_i^2$$

After weighted average just like above, we calculate

$$Impurity\ Reduction = G(Y) - G(Y|X))$$

And one offering highest reduction is chosen as decision variable for splitting.

Classification Error
The formula of leaf node is

$$Classification\ Error = 1 - Max\ p_i$$

Often this is a rarely used one.
Another less heard used ones are
Gain Ratio
The gain ratio "normalizes" the information gain

$$\textbf{Gain Ratio} = \frac{\textbf{Information gain } (\nabla)}{\textbf{Entropy}}$$

Impurity measures such as entropy and Gini Index tend to favor attributes that have large number of distinct values. Therefore Gain Ratio is computed which is used to determine the goodness of a split. Every splitting criterion has their own significance and usage according to their characteristic and attributes type.
Twoing Criteria
The Gini Index may encounter problems when the domain of the target attribute is relatively wide. In this case it is possible to employ binary criterion called twoing criteria. This criterion is defined as:

$$\text{Twoing Criteria } (t) = \frac{P_L P_R (\sum (|p(i/t_L) - p(i/t_R)|))^2}{4}$$

Where, p (i/t) denote the fraction of records belonging to class i at a given node t

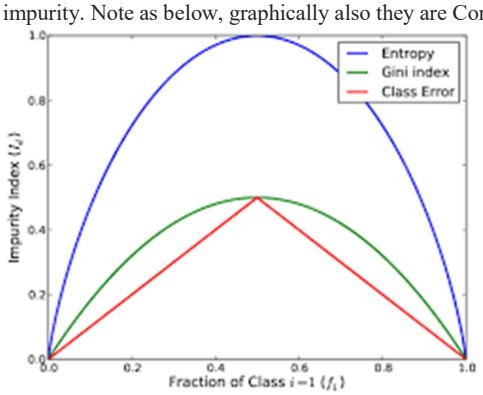
Twoing Criterion for Multiclass Problem

- Classes are numbered {1,2…,j}
- At each node group the classes into two subsets
  - e.g. In the 10 class gym example the two subsets identified by the twoing root node splitter were

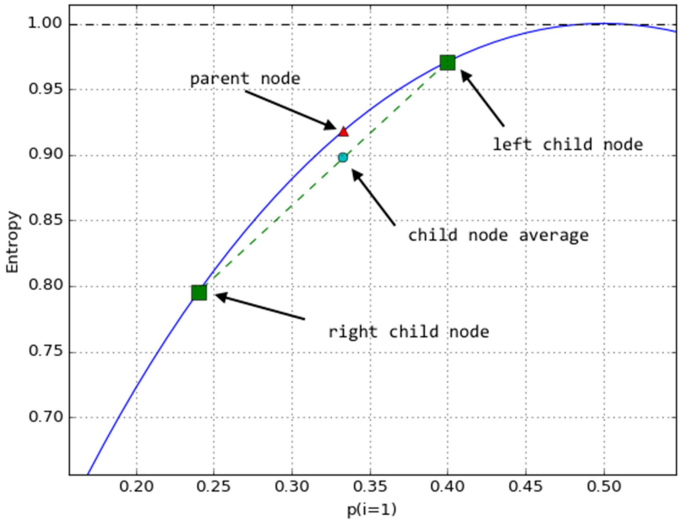
  -C1={1,4,6,7,9,10} C2={2,3,5,8}

- Then the best split for separating these two groups is found
  - The process can be repeated for all possible groupings– best overall is the selected splitter
  - Same as GINI for a binary dependent variable

Little less I could find about it, have a look at this for more understanding.
Highlights:
Binary classification: These are primarily used for Binary split, i.e. two leaf nodes, however when multilevel split is there, we can convert them to Binary split like eg. for color(R, G, B) as R or G, B or G, R or B as splitting decision
Impurity Index(like Information Gain, Gini Index) are concave functions, and we need to maximize the reduction in impurity. Note as below, graphically also they are Convex Functions.



3. Shapes of the above measures: Continuing from above figure the Impurity Index optimize the choice of feature for splitting but following different paths. Note Classification Error gives a straight line curve as opposed to Entropy or Gini Index.

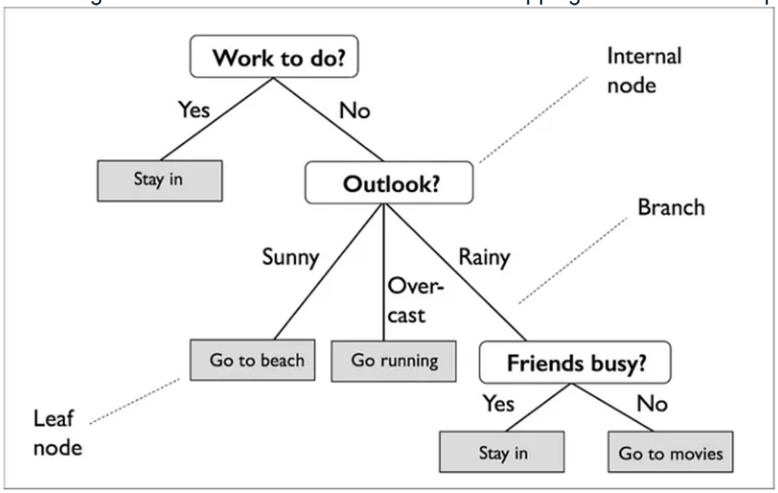


From this figure, we are trying to compare Entropy(or Gini) method with Classification Error. We are trying to compare Impurity before(Red mark)and after(Green dot) splitting. We want the vertical distance between these two points to maximize, so graphically it is quite intuitive that Classification error being straight Line leaves no space between these two points, however Entropy provides greater space over Gini Index.

4. Difference in use when *numerical features instead of categorical:* Categorical is easy to follow, while in Numerical our work is just to find average of two corresponding observations(arranged in ascending) and then check the split's entropy reduction taking each such average as a cutoff. The one providing the max reduction in impurity is chosen as cutoff value.

Regression Based on Decision Trees

Decision Tree Regression is a machine learning technique used to predict continuous numerical values by constructing a tree-like model. It works by splitting data based on features, creating nodes and branching paths until reaching leaf nodes that represent final predictions. These predictions can be the average value of the target variable within that leaf or a function mapping from the feature space to the target value.



Key Concepts:

Tree-like Structure:

The model is structured like a tree, with internal nodes representing features and decision points, and leaf nodes representing predictions.

Splitting Data:

The algorithm splits the data based on features, aiming to create subsets where the target variable has the least variability.

Leaf Node Predictions:

Predictions are made at the leaf nodes, which can be the average target value within that leaf or a function of the features.

How it Works:

1. Data Splitting:
The algorithm starts with a root node containing the entire dataset. It then iteratively splits the data based on the best feature and split point, aiming to maximize the reduction in variance or other suitable impurity measures.

2. Node Creation:
Each split creates new nodes (child nodes) and branches, representing the split conditions.

3. Recursive Splitting:
This process continues recursively until a stopping criterion is met, such as reaching a maximum tree depth or when no further splits significantly improve the model.

4. Leaf Node Predictions:
The final leaf nodes represent the predicted values for the corresponding data subsets.

Benefits:

Interpretability:
Decision trees are relatively easy to understand and interpret, making them useful for explaining prediction models.

Non-linear Relationships:
They can capture non-linear relationships in data, unlike linear regression models.

Feature Importance:
The algorithm can identify the importance of different features in making predictions.

Limitations:

Overfitting:
Decision trees can be prone to overfitting, especially if the tree is too deep or complex. Pruning or using ensemble methods can help mitigate this.

Sensitivity to Data:
They can be sensitive to changes in the data, and the model can be unstable.

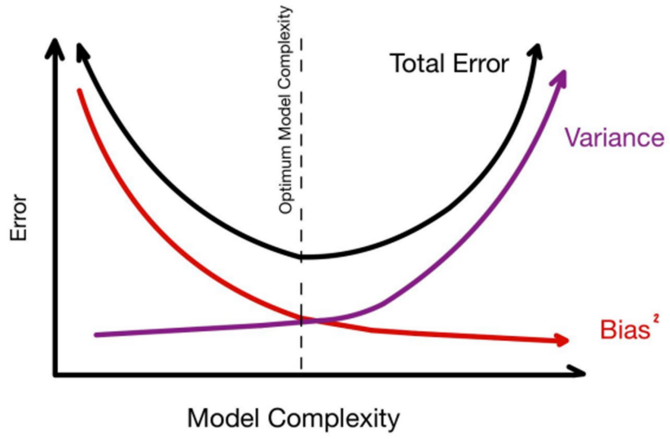
Discrete Output:
While decision trees can be used for regression, they are not ideal for continuous target variables.

Bias–Variance Trade-off

The bias-variance trade-off in machine learning refers to the inherent tension between a model's ability to fit the training data (low bias) and its ability to generalize to unseen data (low variance). The goal is to find a model that strikes a balance between these two, minimizing overall prediction error.

Here's a more detailed explanation:

$$\text{Expected Test Error} = \text{Variance} + \text{Noise} + \text{Bias}^2$$



Understanding Bias and Variance:

Bias:
Refers to the error introduced by making simplifying assumptions about the data or the model's structure. High bias means the model is too simple and cannot capture the underlying patterns in the data, leading to underfitting.

Variance:
Refers to the model's sensitivity to variations in the training data. High variance means the model is too complex and has learned the noise in the training data, leading to overfitting.

The Trade-off:

High Bias, Low Variance:
A model with high bias is simple and consistent, but it may not accurately represent the data.

Low Bias, High Variance:
A model with low bias is complex and can fit the training data well, but it may not generalize well to unseen data.

The Goal:

The goal is to find the sweet spot where the model is complex enough to capture the underlying patterns without overfitting to the noise in the training data.

Why it matters:

Generalization:
The ability of a model to perform well on unseen data is crucial for real-world applications.

Overfitting and Underfitting:
Understanding the bias-variance trade-off helps avoid these common problems in machine learning.

Model Selection:
It guides the choice of model complexity and regularization techniques to achieve optimal performance.

Examples:

Underfitting:
A linear regression model might have high bias and low variance if the relationship between the variables is not linear.

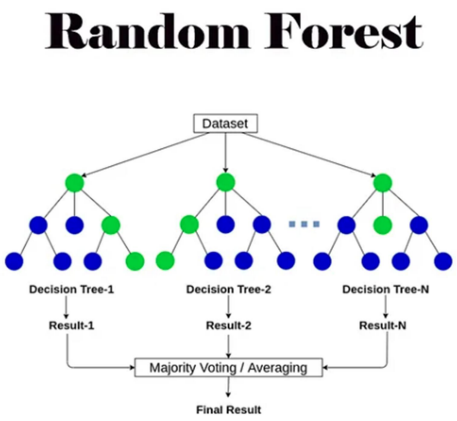
Overfitting:
A complex polynomial regression model might have low bias and high variance if it perfectly fits the training data but performs poorly on new data.

Finding the Balance:
Using techniques like regularization or cross-validation can help find the right balance between bias and variance.

In summary, the bias-variance trade-off is a fundamental concept in machine learning that helps us build models that generalize well to unseen data by finding the optimal balance between model complexity and its ability to fit the training data.

Random Forests for Classification and Regression

Random forests are an ensemble learning method used for both classification and regression tasks, employing multiple decision trees to make predictions, with the final prediction being the mode of the classes (classification) or the average of the predictions (regression).



Random forest — workflow

Here's a more detailed explanation:

What are Random Forests?

Ensemble Method:
Random forests are an ensemble learning method, meaning they combine multiple individual models (decision trees) to make predictions.

Decision Trees:
Each individual model in a random forest is a decision tree.

Randomness:
The "random" in "random forest" refers to the way the decision trees are constructed, with each tree being trained on a random subset of the data and a random subset of the features.

Classification vs. Regression:
Random forests can be used for both classification (predicting categorical outcomes) and regression (predicting continuous outcomes).

How Random Forests Work:

Training:

The algorithm constructs multiple decision trees during training.
Each tree is trained on a random subset of the training data (using bootstrapping) and a random subset of the features.

Prediction:

Classification: For classification tasks, the final prediction is the class that is the mode (most frequent) of the predictions from all the individual trees.
Regression: For regression tasks, the final prediction is the average of the predictions from all the individual trees.

Advantages of Random Forests:

High Accuracy: Random forests are known for their high accuracy and predictive power.
Robustness: They are relatively robust to overfitting and can handle noisy data well.
Feature Importance: Random forests can provide insights into the importance of different features in the data.
Handles Missing Values: Random forests can handle missing values in the data without requiring preprocessing.

Disadvantages of Random Forests:

Computational Cost:
Training random forests can be computationally expensive, especially for large datasets.
Interpretability:
Random forests are often considered "black box" models, meaning it can be difficult to understand why they make certain predictions.
Parameter Tuning:
The performance of random forests can depend on the choice of hyperparameters, which may require tuning.

Applications:

Classification:
- Spam detection.
- Medical diagnosis.
- Customer churn prediction.

Regression:
- Predicting house prices.
- Forecasting stock prices.
- Estimating customer lifetime value.

Introduction to the Bayes Classifier

A Bayesian classifier, or more specifically a Naive Bayes classifier, is a type of probabilistic classifier based on Bayes' theorem. It's a supervised machine learning algorithm used to classify data by assigning it to the most likely class based on its features. The Naive Bayes classifier makes the simplifying assumption that features are conditionally independent, meaning the presence of one feature doesn't affect the presence of another.

Here's a more detailed breakdown:

Bayes' Theorem:
The foundation of this classifier is Bayes' theorem, which describes the probability of an event based on prior knowledge of related conditions.
Probabilistic Approach:
Bayesian classifiers work by estimating the probability of a data point belonging to each class.
Naïve Assumption:
The term "naive" refers to the assumption that all features are independent of each other, given the class. This simplification makes the calculations easier and faster, while still achieving good performance in many cases.
Classification Process:
The classifier calculates the probability of each class given the input data and then assigns the data point to the class with the highest probability.
Applications:
Naive Bayes classifiers are widely used in various applications, including:
- **Text classification:** Identifying the topic or sentiment of a piece of text.
- **Spam detection:** Classifying emails as spam or not spam.
- **Medical diagnosis:** Predicting the likelihood of a disease based on symptoms.
- **Weather prediction:** Predicting the weather conditions based on various factors.

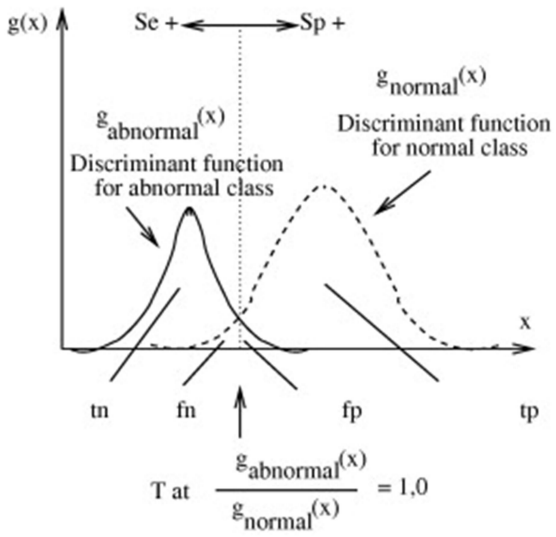
Advantages:
- Simple to implement and understand.
- Requires less training data compared to other classifiers.
- Fast to train and predict.
- Works well with both continuous and categorical data.

Limitations:
- The independence assumption can be unrealistic in some cases.
- Sensitive to irrelevant features.

The Bayes Classifier:

Introduction to the Bayes Classifier

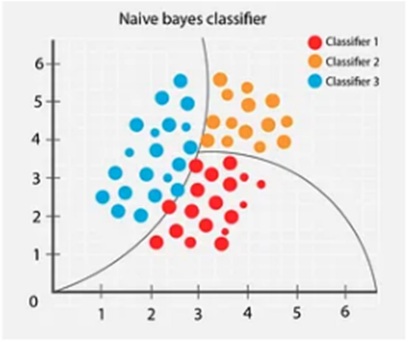
Bayesian classifiers are statistical classifiers, based on Bayes' theorem. They can predict class membership probabilities such as the probability that a given tuple belongs to a particular class. Naive Bayesian classifiers assume that all features in are mutually independent.



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

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$



Bayes' Rule and Bayesian inference are fundamental concepts in probability and statistics, used to update beliefs about an event based on new evidence. Bayes' Rule provides the mathematical framework for this updating, while Bayesian inference is the process of applying that framework to draw conclusions about a model or hypothesis given observed data.
Here's a more detailed explanation:

Bayes' Rule:

Bayes' Rule is a theorem that describes the probability of an event, given that another event has already occurred. It essentially provides a way to revise your prior beliefs about an event in light of new information.
The formula for Bayes' Rule is: P(A|B) = [P(B|A) * P(A)] / P(B).

- **P(A|B):** The posterior probability of event A, given that event B has occurred (your updated belief).
- **P(B|A):** The likelihood of event B, given that event A is true (how likely the evidence is, assuming A is true).
- **P(A):** The prior probability of event A (your initial belief about A).
- **P(B):** The probability of event B (evidence).

In simpler terms, Bayes' Rule tells you how to combine your prior knowledge with the new evidence to arrive at a more informed belief.

Bayesian Inference:

Bayesian inference is a statistical method that uses Bayes' Rule to update beliefs about parameters of a model given observed data.
It involves assigning prior probabilities to possible models or hypotheses, then using the observed data (likelihood) to update these probabilities to posterior probabilities.
The posterior probability distribution represents the updated belief about the model or hypothesis, given the data.
Bayesian inference is widely used in various fields, including machine learning, medical diagnosis, and scientific modeling.

Key Differences:

Bayes' Rule is a mathematical formula. It's a specific equation that relates probabilities.
Bayesian inference is a statistical methodology. It's a process of using Bayes' Rule to draw conclusions from data.

In essence:

Bayes' Rule provides the mathematical framework for updating beliefs.
Bayesian inference is the practical application of that framework to make statistical inferences.

The Bayes Classifier and its Optimality

The Bayes Optimal Classifier (BOC) in machine learning is the theoretically best possible classifier for a given classification problem. It makes the most probable prediction for a new input, minimizing the probability of classification error. While practically unachievable due to unknown class-membership probabilities, it serves as a benchmark for evaluating the performance of other learning algorithms.

Here's a more detailed explanation:

Key Concepts:

- **Bayes Theorem:**
  The BOC is based on Bayes' Theorem, which relates the probabilities of events.
- **Posterior Probability:**
  The BOC makes predictions based on the posterior probability of a class given the input features.
- **Prior Probability:**
  The BOC also considers the prior probability of each class.
- **Maximum A Posteriori (MAP):**
  The BOC is closely related to the MAP principle, where the class with the highest posterior probability is selected.
- **Bayes Error:**
  The BOC achieves the minimum possible error rate, known as the Bayes error.

How it works:

1. **Determine Posterior Probabilities:** For a new input, the BOC calculates the posterior probability of each class (e.g., P(class A | input data)).
2. **Select the Most Probable Class:** The BOC predicts the class with the highest posterior probability.
3. **Theoretical Optimality:** By making predictions based on these probabilities, the BOC minimizes the overall classification error rate.

Why it's important:

- **Theoretical Benchmark:**
  The BOC provides a theoretical upper bound on the performance of any classifier.
- **Evaluation Tool:**
  The BOC is used to evaluate the performance of other learning algorithms, comparing their error rates to the Bayes error.
- **Understanding Learning Algorithms:**
  The BOC helps understand the limitations and potential of different learning algorithms.

Limitations:

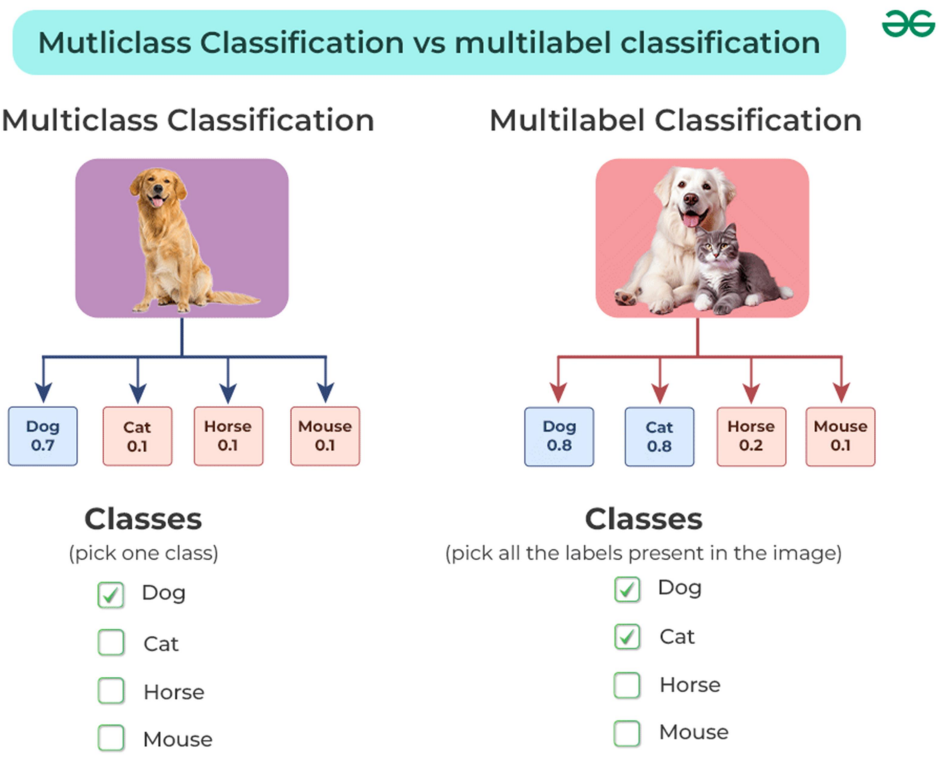
- **Unknown Class Probabilities:**
  The BOC requires knowledge of the class-membership probabilities, which are often unknown in real-world scenarios.
- **Practical Unachievability:**
  Due to the need for perfect knowledge of class probabilities, the BOC is practically unachievable.

In essence, the Bayes Optimal Classifier is a theoretical ideal that helps us understand the limits of what can be achieved in classification and serves as a benchmark for comparing the performance of different machine learning algorithms.

Multi-Class Classification

Multiclass Classification vs Multi-label Classification

Multiclass classification is a machine learning task where the goal is to assign instances to one of multiple predefined classes or categories, where each instance belongs to exactly one class. Whereas multilabel classification is a machine learning task where each instance can be associated with multiple labels simultaneously, allowing for the assignment of multiple binary labels to the instance. In this article we are going to understand the multi-class classification and multi-label classification, how they are different, how they are evaluated, how to choose the best method for your problem, and much more.



What is Multiclass Classification?

Multiclass classification is a machine learning challenge focused on categorizing data into more than two classes. While binary classification involves distinguishing between only two classes, multiclass classification expands this scope to involve distinguishing between multiple classes. In essence, the goal is to train a model that can effectively sort instances into various predefined categories, providing a nuanced solution for scenarios where items can belong to more than two exclusive groups. This approach is commonly employed in tasks such as handwriting recognition, email categorization, and image classification involving more than two distinct categories.

Multiclass classification is a type of machine learning task where the goal is to categorize instances into one of several predefined classes. Unlike binary classification, where there are only two possible outcomes, multiclass classification involves distinguishing between multiple classes or categories. The fundamental idea is to teach a model to assign the most appropriate class label to each instance based on its features.

Multiclass classification finds application in a wide range of real-world scenarios. Consider email categorization, where emails need to be sorted into categories like "spam," "ham" (non-spam), or "important." Another classic example is handwritten digit recognition, where the task is to identify which digit (0 through 9) is written in a given image. Other applications include speech recognition, sentiment analysis, and image classification into multiple categories.

Model Training Techniques:

Training a multiclass classification model involves employing specific techniques to ensure accurate class assignment. One common approach is to use softmax activation in the output layer of the neural network. Softmax converts the raw model outputs into probabilities, assigning higher probabilities to the correct classes. Additionally, categorical cross-entropy loss is often used as the objective function during training. This loss function measures the dissimilarity between the predicted probabilities and the actual class labels, guiding the model to minimize errors and improve accuracy.

Evaluation Metrics:

To assess the performance of a multiclass classification model, various evaluation metrics like accuracy, precision, recall (sensitivity) and F1 score.

Understanding these concepts is crucial for practitioners working on multiclass classification problems, as they form the foundation for designing effective models and assessing their accuracy in real-world applications.

What is Multi-label Classification?

Multi-label classification is a machine learning paradigm where instances can be associated with multiple labels simultaneously. Unlike traditional classification tasks, where an instance is assigned a single exclusive label,

multi-label classification recognizes the possibility for instances to exhibit characteristics that span across various categories. The goal is to develop models capable of accurately predicting and assigning a set of relevant labels to each instance, reflecting the complex relationships and diversity inherent in real-world datasets. This approach acknowledges the overlapping nature of labels, providing a more realistic representation of the multifaceted attributes present in the data.

Multi-label classification is a machine learning task where instances can be associated with multiple labels simultaneously. This differs from multiclass classification, where each instance is assigned to one and only one class. In multi-label scenarios, an instance may exhibit characteristics that correspond to several different categories, making the task more intricate and reflecting the complexity often found in real-world data.

Multi-label classification is highly applicable in diverse scenarios where instances can possess multiple attributes or labels. Examples include:

- Document Tagging: Assigning multiple tags or topics to a document, such as labeling an article as both "technology" and "business."
- Image Classification with Multiple Labels: Identifying and labeling multiple objects or features within an image, like recognizing both "cat" and "outdoor" in a photograph.

Model Training Techniques:

Training models for multi-label classification involves specific techniques to accommodate the simultaneous assignment of multiple labels to instances:

- Sigmoid Activation: In the output layer of the neural network, sigmoid activation is often used. Unlike softmax in multiclass scenarios, sigmoid independently activates each output node, producing a value between 0 and 1, representing the likelihood of the corresponding label being present.
- Binary Cross-Entropy Loss: This loss function is employed during training to measure the dissimilarity between the predicted probabilities and the actual presence or absence of each label. It guides the model to minimize errors in its multi-label predictions.

Evaluation Metrics:

Assessing the performance of a multi-label classification model requires specific metrics tailored to handle the complexity of multiple labels per instance:

- Hamming Loss: This metric calculates the fraction of labels that are incorrectly predicted. It provides a comprehensive measure of overall model performance in terms of label accuracy.
- Precision at k: Precision at k evaluates the precision of the top-k predicted labels, recognizing that not all labels need to be considered. It accounts for scenarios where only the most relevant labels are of interest.
- Recall at k: Similar to precision at k, recall at k assesses the recall of the top-k predicted labels. It focuses on capturing the relevant labels among the top predictions.

Understanding these nuances of multi-label classification is essential for practitioners working on tasks where instances can belong to multiple categories simultaneously, ensuring effective model design and evaluation in complex real-world scenarios.

Differences between Multi class and Multi label Classification

Features	Multi class classification.	Multi label classification
Output Structure:	The output is a single class label assigned to each instance, indicating the most probable or correct class.	The output is a set of binary values indicating the presence or absence of each label for each instance. Instances can be associated with multiple labels simultaneously.
Model Output:	the model assigns a single class label to each instance based on the class with the highest probability or confidence.	The model outputs a binary vector for each instance, where each element corresponds to a label, indicating whether it is present or not.
Training Techniques:	Techniques like softmax activation and categorical cross-entropy loss are commonly used for training models to handle multiple classes.	Techniques like sigmoid activation and binary cross-entropy loss are employed, treating each label independently.

Features	Multi class classification.	Multi label classification
Class Assignment:	Each instance is assigned to one and only one class, making the classification mutually exclusive.	Instances can be associated with multiple labels, allowing for overlapping or shared characteristics.
Evaluation Metrics:	Metrics such as accuracy, precision, recall, and F1 score are commonly used to assess the overall performance of the model.	Metrics like Hamming loss, precision at k, and recall at k are more appropriate, as they account for the presence of multiple labels for each instance.
Model Complexity:	Generally considered simpler as it involves assigning instances to exclusive classes.	Can be more complex due to the need to capture dependencies and correlations between multiple labels.
Problem Complexity:	Typically used for simpler problems where instances belong to mutually exclusive categories.	Suited for more complex scenarios where instances can exhibit characteristics of multiple labels simultaneously.

Choosing Between Multi-Class and Multi-Label Classification

When embarking on a classification task, one of the foundational decisions is whether to opt for multi-class or multi-label classification, and this choice significantly influences the model's performance and relevance to real-world scenarios.

- Assess whether the instances in your dataset belong to mutually exclusive classes (Multi-Class) or if they can have multiple labels simultaneously (Multi-Label). Understanding the nature of labels is fundamental in choosing the appropriate classification approach.
- Examine the relationships between labels. If the labels are independent or weakly correlated, multi-class classification may be suitable. For strong correlations or overlapping characteristics, multi-label classification is more appropriate.
- Gauge the complexity of your classification problem. Multi-class classification is generally simpler as it deals with exclusive categorization. If the problem is inherently complex and instances can have diverse characteristics, opt for multi-label classification.
- Consider domain-specific requirements and constraints. Some domains naturally lend themselves to one approach over the other based on the inherent characteristics of the data and the specific objectives of the task.

In conclusion, the choice between multi-class and multi-label classification should be made considering the intricacies of the problem, the nature of the data, and the specific requirements of the application. Each approach has its merits, and selecting the most suitable classification method is pivotal for achieving optimal model performance in diverse real-world scenarios.

Class Conditional Independence and Naive Bayes Classifier (NBC)

Naive Bayes Classifiers

Naive Bayes is a classification algorithm that uses probability to predict which category a data point belongs to, assuming that all features are unrelated. This article will give you an overview as well as more advanced use and implementation of Naive Bayes in machine learning.

Illustration behind the Naive Bayes algorithm. We estimate $P(x_\alpha|y)$ *independently in each dimension (middle two images) and then obtain an estimate of the full data distribution by assuming conditional independence* $P(x|y)=\prod_\alpha P(x_\alpha|y)$ *(very right image).*

Key Features of Naive Bayes Classifiers

The main idea behind the Naive Bayes classifier is to use **Bayes' Theorem** to classify data based on the probabilities of different classes given the features of the data. It is used mostly in high-dimensional text classification

- The Naive Bayes Classifier is a simple probabilistic classifier and it has very few number of parameters which are used to build the ML models that can predict at a faster speed than other classification algorithms.
- It is a probabilistic classifier because it assumes that one feature in the model is independent of existence of another feature. In other words, each feature contributes to the predictions with no relation between each other.
- Naïve Bayes Algorithm is used in spam filtration, Sentimental analysis, classifying articles and many more.

Why it is Called Naive Bayes?

It is named as "Naive" because it assumes the presence of one feature does not affect other features. The "Bayes" part of the name refers to its basis in Bayes' Theorem.

Consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each tuple classifies the conditions as fit("Yes") or unfit("No") for playing golf. Here is a tabular representation of our dataset.

	Outlook	Temperature	Humidity	Windy	Play Golf
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes

	Outlook	Temperature	Humidity	Windy	Play Golf
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

The dataset is divided into two parts, namely, **feature matrix** and the **response vector**.

- Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of **dependent features**. In above dataset, features are 'Outlook', 'Temperature', 'Humidity' and 'Windy'.
- Response vector contains the value of **class variable**(prediction or output) for each row of feature matrix. In above dataset, the class variable name is 'Play golf'.

Assumption of Naive Bayes

The fundamental Naive Bayes assumption is that each feature makes an:

- **Feature independence:** This means that when we are trying to classify something, we assume that each feature (or piece of information) in the data does not affect any other feature.
- **Continuous features are normally distributed:** If a feature is continuous, then it is assumed to be normally distributed within each class.
- **Discrete features have multinomial distributions:** If a feature is discrete, then it is assumed to have a multinomial distribution within each class.
- **Features are equally important:** All features are assumed to contribute equally to the prediction of the class label.
- **No missing data:** The data should not contain any missing values.

Introduction to Bayes' Theorem

Bayes' Theorem provides a principled way to reverse conditional probabilities. It is defined as:

$P(y|X) = P(X|y) \cdot P(y) P(X) P(y|X) = P(X) P(X|y) \cdot P(y)$

Where:

- P(y|X)$P(y|X)$: Posterior probability, probability of class yy given features XX
- P(X|y)$P(X|y)$: Likelihood, probability of features XX given class yy
- P(y)$P(y)$: Prior probability of class yy
- P(X)$P(X)$: Marginal likelihood or evidence

Naive Bayes Working

1. Terminology

Consider a classification problem (like predicting if someone plays golf based on weather). Then:

- yy is the class label (e.g. "Yes" or "No" for playing golf)
- X=(x1,x2,...,xn)$X = (x1, x2, ..., xn)$ is the feature vector (e.g. Outlook, Temperature, Humidity, Wind)

A sample row from the dataset:

$X = (Rainy, Hot, High, False), y = No$ $X = (Rainy, Hot, High, False), y = No$

This represents:

What is the probability that someone will not play golf given that the weather is Rainy, Hot, High humidity, and No wind?

2. The Naive Assumption

The "naive" in Naive Bayes comes from the assumption that all features are independent given the class. That is:

$P(x1, x2, ..., xn|y) = P(x1|y) \cdot P(x2|y) \cdots P(xn|y) P(x1, x2, ..., xn|y) = P(x1|y) \cdot P(x2|y) \cdots P(xn|y)$

Thus, Bayes' theorem becomes:

$P(y|x1, ..., xn) = P(y) \cdot \prod_{i=1}^{n} P(xi|y) P(x1) P(x2) ... P(xn) P(y|x1, ..., xn) = P(x1) P(x2) ... P(xn) P(y) \cdot \prod_{i=1}^{n} P(xi|y)$

Since the denominator is constant for a given input, we can write:
$P(y|x1,...,xn) \propto P(y) \cdot \prod_{i=1} nP(xi|y) P(y|x1,...,xn) \propto P(y) \cdot \prod_{i=1} nP(xi|y)$

3. Constructing the Naive Bayes Classifier

We compute the posterior for each class y*y* and choose the class with the highest probability:
$y\hat{} = arg \quad max \quad yP(y) \cdot \prod_{i=1} nP(xi|y) y\hat{} = argmaxyP(y) \cdot \prod_{i=1} nP(xi|y)$
This becomes our Naive Bayes classifier.

4. Example: Weather Dataset

Let's take a dataset used for predicting if golf is played based on:

- **Outlook**: Sunny, Rainy, Overcast
- **Temperature**: Hot, Mild, Cool
- **Humidity**: High, Normal
- **Wind**: True, False

Outlook

	Yes	No	P(yes)	P(no)
Sunny	3	2	3/10	2/4
Overcast	4	0	4/10	0/4
Rainy	3	2	3/10	2/4
Total	10	4	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind

	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play		P(Yes)/P(No)
Yes	9	9/14
No	5	5/14
Total	14	100%

Example Tables for Naive Bayes

Example Input: X=(Sunny,Hot,Normal,False)X=(*Sunny*,*Hot*,*Normal*,*False*)
Goal: Predict if golf will be played (Yes or No).

5. Pre-computation from Dataset

Class Probabilities:

From dataset of 14 rows:

- P(Yes)=914P(Yes)=149
- P(No)=514P(No)=145

Conditional Probabilities (Tables 1–4):

Feature	Value	P (Value | Yes)	P (Value | No)
Outlook	Sunny	2/9	3/5
Temperature	Hot	2/9	2/5
Humidity	Normal	6/9	1/5
Wind	False	6/9	2/5

6. Calculate Posterior Probabilities

For Class = Yes:

$P(Yes \mid today) \propto 29 \cdot 29 \cdot 69 \cdot 69 \cdot 914 P(Yes \mid today) \propto 92 \cdot 92 \cdot 96 \cdot 96 \cdot 149$

$P(Yes \mid today) \approx 0.02116 P(Yes \mid today) \approx 0.02116$

For Class = No:

$P(No \mid today) \propto 35 \cdot 25 \cdot 15 \cdot 25 \cdot 514 P(No \mid today) \propto 53 \cdot 52 \cdot 51 \cdot 52 \cdot 145$

$P(No \mid today) \approx 0.0068 P(No \mid today) \approx 0.0068$

7. Normalize Probabilities

To compare:

$P(Yes \mid today) = 0.021160.02116 + 0.0068 \approx 0.756 P(Yes \mid today) = 0.02116 + 0.00680.02116 \approx 0.756$

$P(No \mid today) = 0.00680.02116 + 0.0068 \approx 0.244 P(No \mid today) = 0.02116 + 0.00680.0068 \approx 0.244$

8. Final Prediction

Since:

$P(Yes \mid today) > P(No \mid today) P(Yes \mid today) > P(No \mid today)$

The model predicts: ***Yes (Play Golf)***

Naive Bayes for Continuous Features

For continuous features, we assume a Gaussian distribution:

$P(xi \mid y) = 12\pi\sigma y2 \exp \quad (-(xi-\mu y)22\sigma y2) P(xi \mid y) = 2\pi\sigma y21 \exp(-2\sigma y2(xi-\mu y)2)$

Where:

- μyμy is the mean of feature xixi for class yy
- σy2$\sigma y2$ is the variance of feature xixi for class yy

This leads to what is called **Gaussian Naive Bayes**.

Types of Naive Bayes Model

There are three types of Naive Bayes Model :

1. Gaussian Naive Bayes

In **Gaussian Naive Bayes**, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown below:

2. Multinomial Naive Bayes

Multinomial Naive Bayesis used when features represent the frequency of terms (such as word counts) in a document. It is commonly applied in text classification, where term frequencies are important.

3. Bernoulli Naive Bayes

Bernoulli Naive Bayes deals with binary features, where each feature indicates whether a word appears or not in a document. It is suited for scenarios where the presence or absence of terms is more relevant than their frequency. Both models are widely used in document classification tasks

Advantages of Naive Bayes Classifier

- Easy to implement and computationally efficient.
- Effective in cases with a large number of features.
- Performs well even with limited training data.
- It performs well in the presence of categorical features.
- For numerical features data is assumed to come from normal distributions

Disadvantages of Naive Bayes Classifier

- Assumes that features are independent, which may not always hold in real-world data.
- Can be influenced by irrelevant attributes.
- May assign zero probability to unseen events, leading to poor generalization.

Applications of Naive Bayes Classifier

- **Spam Email Filtering**: Classifies emails as spam or non-spam based on features.
- **Text Classification**: Used in sentiment analysis, document categorization, and topic classification.
- **Medical Diagnosis:** Helps in predicting the likelihood of a disease based on symptoms.
- **Credit Scoring:** Evaluates creditworthiness of individuals for loan approval.
- **Weather Prediction**: Classifies weather conditions based on various factors.

Linear Discriminants for Machine Learning

Introduction to Linear Discriminants

Linear Discriminants for Classification

When working with high-dimensional datasets it is important to apply dimensionality reduction techniques to make data exploration and modeling more efficient. **Linear Discriminant Analysis (LDA)** also known as Normal Discriminant Analysis **is supervised classification problem that helps separate two or more classes by converting higher-dimensional data space into a lower-dimensional space. I**t is used to identify a linear combination of features that best separates classes within a dataset.

2 Classes overlapping

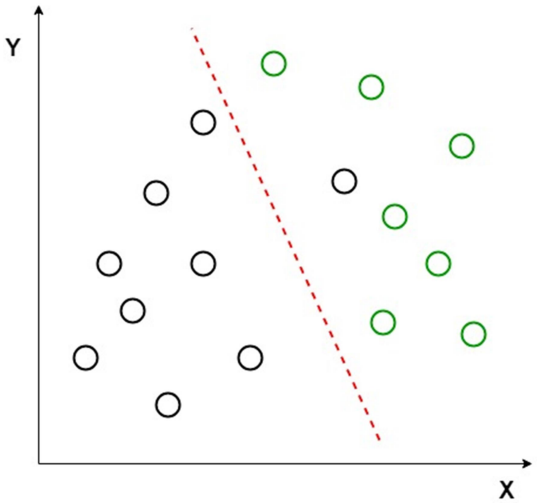
For example we have two classes that need to be separated efficiently. Each class may have multiple features and using a single feature to classify them may result in overlapping. To solve this **LDA is used as it uses multiple features to improve classification accuracy.** LDA works by some assumptions and we are required to understand them so that we have a better understanding of its working.

Key Assumptions of LDA

For LDA to perform effectively, certain assumptions are made:

- **Gaussian Distribution:** The data in each class should follow a normal bell-shaped distribution.
- **Equal Covariance Matrices:** All classes should have the same covariance structure.
- **Linear Separability:** The data should be separable using a straight line or plane.

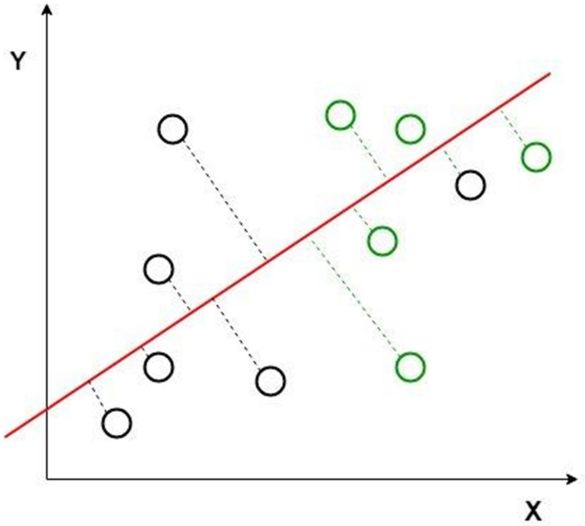
If these assumptions are met LDA can produce very good results. For example when data points belonging to two classes are plotted if they are not linearly separable LDA will attempt to find a projection that maximizes class separability.



Linearly Separable Dataset

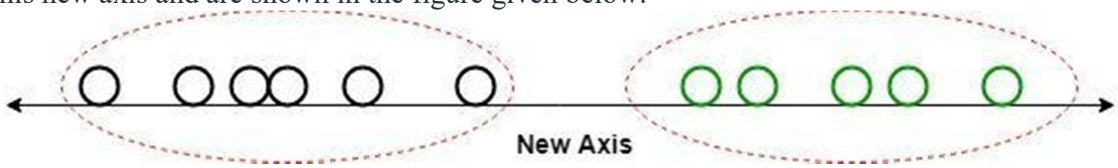
Image shows an example where the classes (black and green circles) are not linearly separable. LDA attempts to separate them using red dashed line. **It uses both axes (X and Y) to generate a new axis in such a way that it maximizes the distance between the means of the two classes while minimizing the variation within each class.** This transforms the

dataset into a space where the classes are better separated. After transforming the data points along a new axis LDA maximizes the class separation. This new axis allows for clearer classification by projecting the data along a line that enhance the distance between the means of the two classes.



The perpendicular distance between the line and points

Perpendicular distance between the decision boundary and the data points helps us to isualize how LDA works by reducing class variation and increasing separability. After generating this new axis using the above-mentioned criteria all the data points of the classes are plotted on this new axis and are shown in the figure given below.



LDA

It shows how LDA creates a new axis to project the data and separate the two classes effectively along a linear path. But it fails when the mean of the distributions are shared as it becomes impossible for LDA to find a new axis that makes both classes linearly separable. In such cases we use non-linear discriminant analysis.

How does LDA work

LDA works by finding directions in the feature space that best separate the classes. It does this by maximizing the difference between the class means while minimizing the spread within each class.

Let's assume we have two classes with d-dimensional samples such as x1,x2,...xn*x*1,*x*2,...*xn* where:

- n1*n*1 samples belong to class c1*c*1
- n2*n*2 samples belong to class c2*c*2.

If xi*xi* represents a data point its projection onto the line represented by the unit vector v is vTxi*vTxi*. Let the means of class c1*c*1 and class c2*c*2 before projection be μ1 and μ2 respectively. After projection the new means are μ^1=vTμ1*μ*^1=*vTμ*1and μ^2=vTμ2*μ*^2 =*vTμ*2.

Our aim to normalize the difference |μ^1−μ^2||*μ*^1−*μ*^2|to maximize the class separation. The scatter for samples of class c1*c*1 is calculated as:

$$s12=\sum xi\in c1(xi-\mu 1)2 s12=\sum xi\in c1(xi-\mu 1)2$$

Similarly for class c2*c*2:

$$s22=\sum xi\in c2(xi-\mu 2)2 s22=\sum xi\in c2(xi-\mu 2)2$$

The goal is to maximize the ratio of the between-class scatter to the within-class scatter, which leads us to the following criteria:

$$J(v)=|\mu\hat{}1-\mu\hat{}2|s12+s22 J(v)=s12+s22|\mu\hat{}1-\mu\hat{}2|$$

For the best separation we calculate the eigenvector corresponding to the highest eigenvalue of the scatter matrices $sw-1sbsw-1sb$.

Extensions to LDA

1. **Quadratic Discriminant Analysis (QDA):** Each class uses its own estimate of variance (or covariance) allowing it to handle more complex relationships.
2. **Flexible Discriminant Analysis (FDA):** Uses non-linear combinations of inputs such as splines to handle non-linear separability.
3. **Regularized Discriminant Analysis (RDA):** Introduces regularization into the covariance estimate to prevent overfitting.

Implementation of LDA using Python

In this implementation we will perform linear discriminant analysis using Scikit-learn library on the **Iris dataset**.

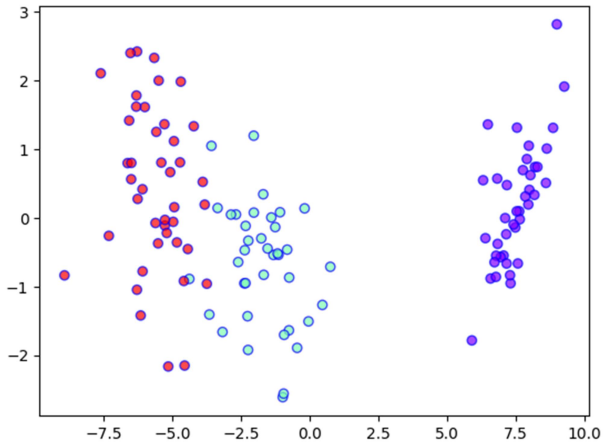
- **StandardScaler()**: Standardizes the features to ensure they have a mean of 0 and a standard deviation of 1 removing the influence of different scales.
- **fit_transform()**: Standardizes the feature data by applying the transformation learned from the training data ensuring each feature contributes equally.
- **LabelEncoder()**: Converts categorical labels into numerical values that machine learning models can process.
- **fit_transform() on y**: Transforms the target labels into numerical values for use in classification models.
- **LinearDiscriminantAnalysis()**: Reduces the dimensionality of the data by projecting it into a lower-dimensional space while maximizing the separation between classes.
- **transform() on X_test**: Applies the learned LDA transformation to the test data to maintain consistency with the training data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

iris = load_iris()
dataset = pd.DataFrame(columns=iris.feature_names,
                       data=iris.data)
dataset['target'] = iris.target

X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 4].values

sc = StandardScaler()
X = sc.fit_transform(X)
le = LabelEncoder()
y = le.fit_transform(y)
```

```
X_train, X_test,\
    y_train, y_test = train_test_split(X, y,
                                       test_size=0.2)

lda = LinearDiscriminantAnalysis(n_components=2)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)

plt.scatter(
    X_train[:, 0], X_train[:, 1],
    c=y_train,
    cmap='rainbow',
    alpha=0.7, edgecolors='b'
)

classifier = RandomForestClassifier(max_depth=2,
                                    random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

print('Accuracy : ' + str(accuracy_score(y_test, y_pred)))
conf_m = confusion_matrix(y_test, y_pred)
print(conf_m)
```

Output:

Accuracy : 0.9
[[8 0 0]
[0 8 2]
[0 1 11]]



Scatter plot of the iris data mapped into 2D

This scatter plot shows three distinct groups of data points, represented by different colors. The group on the right (dark blue) is clearly separated from the others indicate it's very different. The other two groups (red and light blue) are positioned closer together with some overlap and suggest they are more similar and harder to separate.

Advantages of LDA

- Simple and computationally efficient.

- Works well even when the number of features is much larger than the number of training samples.
- Can handle multicollinearity.

Disadvantages of LDA

- Assumes Gaussian distribution of data which may not always be the case.
- Assumes equal covariance matrices for different classes which may not hold in all datasets.
- Assumes linear separability which is not always true.
- May not always perform well in high-dimensional feature spaces.

Applications of LDA

1. **Face Recognition**: It is used to reduce the high-dimensional feature space of pixel values in face recognition applications helping to identify faces more efficiently.
2. **Medical Diagnosis**: It classifies disease severity in mild, moderate or severe based on patient parameters helping in decision-making for treatment.
3. **Customer Identification**: It can help identify customer segments most likely to purchase a specific product based on survey data.

Perceptron Classifier

A perceptron is a fundamental type of binary classifier in machine learning, particularly in the context of artificial neural networks. It's a single-layer neural network that takes inputs, applies weights to them, and then uses an activation function (like the step function) to produce a binary output (usually 0 or 1). The perceptron algorithm is used for supervised learning of binary classifiers, meaning it learns to classify data points into one of two classes.

Key Concepts:

- **Binary Classification:**
  Perceptrons are designed to classify data into two distinct categories, often represented as 0 and 1, or -1 and 1.
- **Linear Separability:**
  Perceptrons are effective for data that can be separated by a straight line (or a hyperplane in higher dimensions).
- **Weights and Bias:**
  The algorithm learns weights associated with each input feature and a bias term to make predictions.
- **Activation Function:**
  The activation function (e.g., step function, sigmoid) transforms the weighted sum of inputs into the final binary output.
- **Perceptron Learning Rule:**
  The algorithm iteratively adjusts the weights and bias to minimize the error between the predicted and actual outputs.

  How it Works:

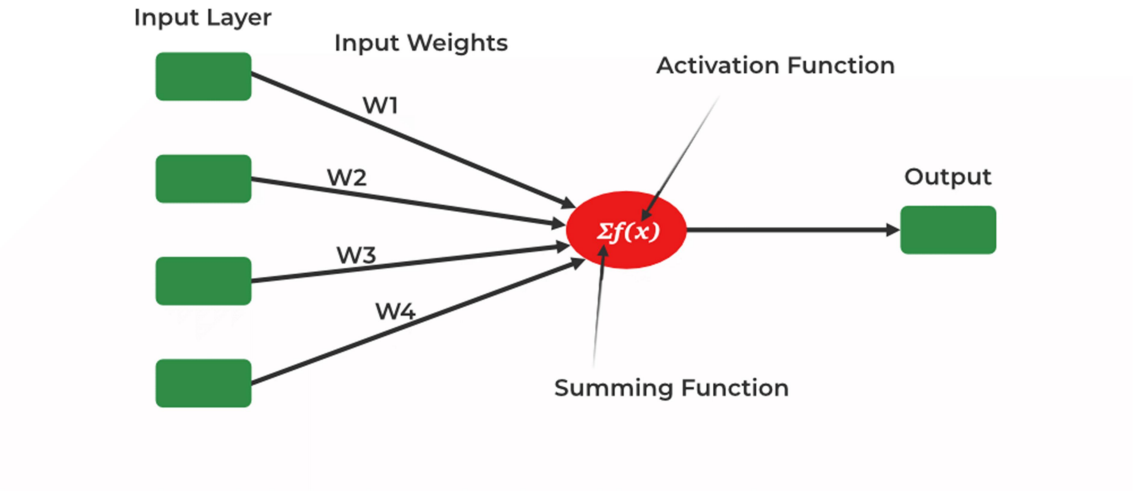
  1. **Input:** The perceptron receives a vector of input features.
  2. **Weighted Sum:** Each input is multiplied by its corresponding weight, and the products are summed together.
  3. **Bias:** A bias term is added to the weighted sum.
  4. **Activation Function:** The sum (along with the bias) is passed through an activation function, which produces a binary output (e.g., 0 or 1).
  5. **Learning:** The perceptron compares its output with the desired output and adjusts the weights and bias accordingly to minimize the error.

  Limitations:

- **Limited to Linear Boundaries:**

Perceptrons can only separate data linearly, meaning they can't handle complex, non-linear relationships.

- **Sensitivity to Data Noise:**
  The perceptron algorithm can be sensitive to noise in the training data, potentially leading to poor generalization.



Perceptron Learning Algorithm

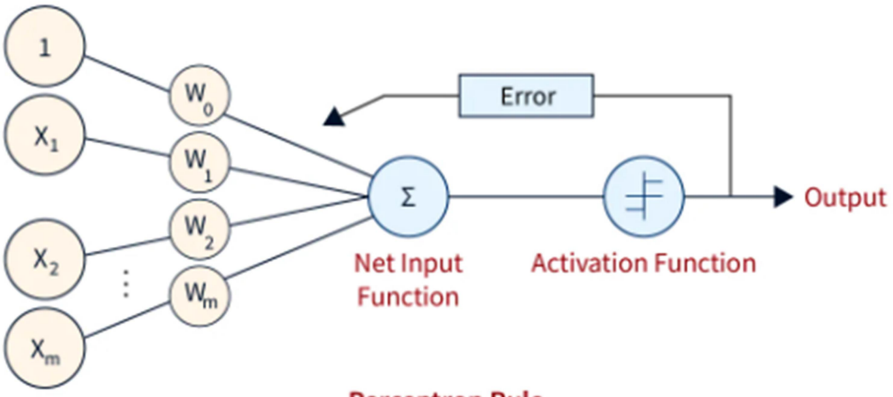
Perceptron Learning Algorithm is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence. The perceptron learning algorithm is treated as the most straightforward Artificial Neural network. It is a supervised learning algorithm of binary classifiers. Hence, it is a single-layer neural network with four main parameters, **i.e.**, input values, weights and Bias, net sum, and an activation function.

What is the Perceptron Learning Algorithm?

There are four significant steps in a perceptron learning algorithm:

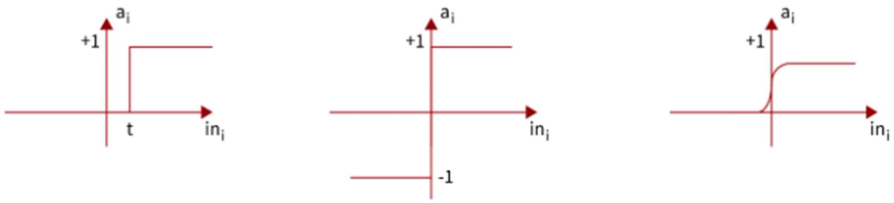
1. First, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows: $\sum wi*xi=x1*w1+x2*w2+\ldots+wn*xn$ $\sum w_i*x_i=x_1*w_1+x_2*w_2+\ldots+w_n*x_n$. Add another essential term called bias 'b' to the weighted sum to improve the model performance. $\sum wi*xi+b$ $\sum w_i*x_i+b$.
2. Next, an activation function is applied to this weighed sum, producing a binary or a continuous-valued output. $Y=f(\sum wi*xi+b)$ $Y=f(\sum w_i*x_i+b)$

3. Next, the difference between this output and the actual target value is computed to get the error term, E, generally in terms of mean squared error. The steps up to this form the forward propagation part of the algorithm. $E=(Y-Y_{actual})2$
4. We optimize this error (loss function) using an optimization algorithm. Generally, some form of gradient descent algorithm is used to find the optimal values of the hyperparameters like learning rate, weight, Bias, etc. This step forms the backward propagation part of the algorithm.



Perceptron Rule

1. **Input Nodes or Input Layer:** Primary component of Perceptron learning algorithm, which accepts the initial input data into the model. Each input node contains an actual value.
2. **Weight and Bias:** The weight parameter represents the strength of the connection between units. Bias can be considered as the line of intercept in a linear equation.
3. **Activation Function:** Final and essential components help determine whether the neuron will fire. The activation function can be primarily considered a step function. There are various types of activation functions used in a perceptron learning algorithm. Some of them are the sign function, step function, sigmoid function, etc.

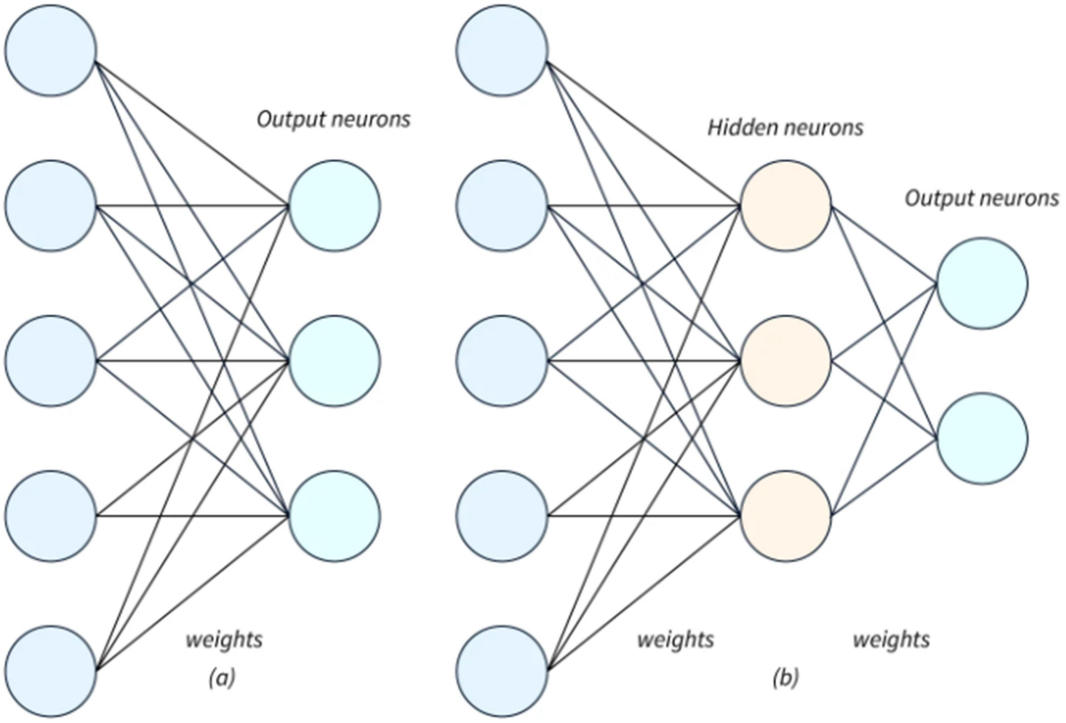


Step Function Sign Function Sigmoid Function

Types of Perceptron Models

Based on the number of layers, perceptrons are broadly classified into two major categories:

1. **Single Layer Perceptron Model:**
   It is the simplest Artificial Neural Network (ANN) model. A single-layer perceptron model consists of a feed-forward network and includes a threshold transfer function for thresholding on the Output. The main objective of the single-layer perceptron model is to classify linearly separable data with binary labels.
2. **Multi-Layer Perceptron Model:**
   The multi-layer perceptron learning algorithm has the same structure as a single-layer perceptron but consists of an additional one or more hidden layers, unlike a single-layer perceptron, which consists of a single hidden layer. The distinction between these two types of perceptron models is shown in the Figure below.



Perceptron Function

Perceptron learning algorithm function $f(x)$ is represented as the product of the input vector (x) and the learned weight vector (w). In mathematical notion, it can be described as:

$f(x)=1, if w.x+b>0 f(x)=0, otherwise$

Where–

- w represents the weight vector which consists of a set of real-valued weights.
- b represents the bias vector.
- x represents the input vector which consists of the input feature values.

Geometry of the Solution Space

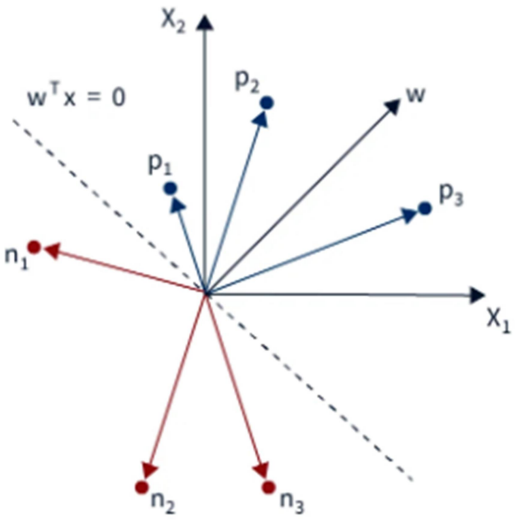
In the previous section, we learned about the weight update rules for the perceptron learning algorithm. We have already established that when x belongs to P, we want $w.x > 0$. That means that the angle between w and x should be less than 90 because the cosine of the slope is proportional to the dot product.

$\cos \alpha = wTx \|w\| \|x\| \cos \alpha \propto wTx \cos\alpha = \|\mathbf{w}\| \|\mathbf{x}\| \mathbf{w}_T\mathbf{x} |\cos\alpha \propto \mathbf{w}_T\mathbf{x}$
$So if wTx>0 \Rightarrow \cos \alpha > 0 \Rightarrow \alpha < 90 So if \mathbf{w}_T\mathbf{x} > 0 \Rightarrow \cos\alpha > 0 \Rightarrow \alpha < 90$

Similarly,

$if wTx<0 \Rightarrow \cos \alpha < 0 \Rightarrow \alpha > 90 if \mathbf{w}_T\mathbf{x} < 0 \Rightarrow \cos\alpha < 0 \Rightarrow \alpha > 90$

So whatever the w vector may be, as long as it makes an angle less than 90 degrees with the positive example data vectors ($x \in\in P$) and an angle more than 90 degrees with the negative example data vectors ($x \in\in N$), we are cool. So ideally, it should look something like this:



x_0 is always 1 so we ignore it for now.

So the angle between w and x should be less than 90 when x belongs to the P class, and the angle between them should be more than 90 when x belongs to the N class. Pause and convince yourself that the above statements are true and you believe them.

Here's Why the Update Works:

(αnew) when $wnew = w + x \cos(\alpha new) \propto wnew\ Tx \propto (w+x)Tx \propto wTx + xTx \propto \cos\alpha + xTx \cos(\alpha new) > \cos\alpha (\alpha new)$ when $wnew = w - x \cos(\alpha new) \propto wnew\ Tx \propto (w-x)Tx \propto wTx - xTx \propto \cos\alpha - xTx \cos(\alpha new) < \cos\alpha (\alpha_{new})$ when $\cos(\alpha_{new})\cos(\alpha_{new})\mathbf{w}_{new} = \mathbf{w} + \mathbf{x} \propto \mathbf{w}_{new}\ T\mathbf{x} \propto (\mathbf{w}+\mathbf{x})T\mathbf{x} \propto \mathbf{w}T\mathbf{x} + \mathbf{x}T\mathbf{x} \propto \cos\alpha + \mathbf{x}T\mathbf{x} > \cos\alpha (\alpha_{new})$ when $\cos(\alpha_{new})\cos(\alpha_{new})\mathbf{w}_{new} = \mathbf{w} - \mathbf{x} \propto \mathbf{w}_{new}\ T\mathbf{x} \propto (\mathbf{w}-\mathbf{x})T\mathbf{x} \propto \mathbf{w}T\mathbf{x} - \mathbf{x}T\mathbf{x} \propto \cos\alpha - \mathbf{x}T\mathbf{x} < \cos\alpha$

So when we are adding x to w, which we do when x belongs to P and $w.x < 0$ (**Case 1**), we are essentially increasing the cos(alpha) value, which means we are decreasing the alpha value, the angle between w and x, which is what we desire. And the similar intuition works for the case when x belongs to N and $w.x \geq 0$ (**Case 2**).

Perceptron Learning Algorithm: Implementation of AND Gate

The steps for this implementation are as follows:

1. **Import all the required libraries:**

```
#import required library
import tensorflow as tf
```

2. **Define Vector Variables for Input and Output:**

```
#input1, input2 and bias
train_in = [
    [1., 1.,1],
    [1., 0,1],
    [0, 1.,1],
    [0, 0,1]]

#output
train_out = [
[1.],
[0],
[0],
[0]]
```

3. **Define the Weight Variable:**

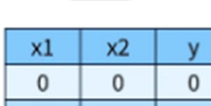
```
#weight variable initialized with random values using
random_normal()
w = tf.Variable(tf.random_normal([3, 1], seed=12))
```

4. **Define placeholders for Input and Output:**

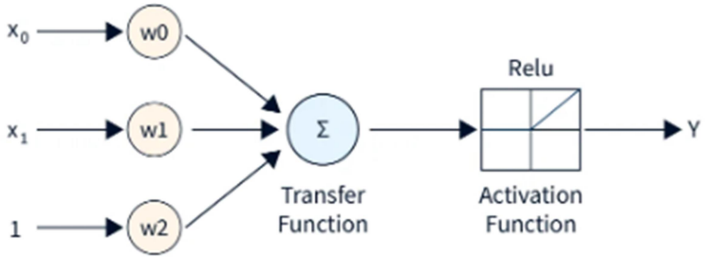
```
#Placeholder for input and Output
x = tf.placeholder(tf.float32,[None,3])
y = tf.placeholder(tf.float32,[None,1])
```

5. **Calculate Output and Activation Function:**

AND Gate



x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1




```
#calculate output
output = tf.nn.relu(tf.matmul(x, w))
```

6. **Calculate the Cost or Error:**

```
#Mean Squared Loss or Error
loss = tf.reduce_sum(tf.square(output - y))
```

7. **Minimize Error:**

```
#Minimize loss using GradientDescentOptimizer with a learning
rate of 0.01
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

8. **Initialize all the variables:**

```
#Initialize all the global variables
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

9. **Training Perceptron learning algorithm in Iterations:**

```
training_epochs = 1000

#Compute cost w.r.t to input vector for 1000 epochs

for epoch in range(training epochs):
    sess.run(train, {x:train_in,y:train_out})
    cost = sess.run(loss,feed dict={x:train in,y:train out})
    if i > 990:
        print('Epoch--',epoch,'--loss--',cost)
```

Output:

Following is the final Output obtained after my perceptron model has been trained.

```
Epoch-- 991 --loss-- 0.0003835174
Epoch-- 992 --loss-- 0.00038088957
Epoch-- 993 --loss-- 0.0003782803
Epoch-- 994 --loss-- 0.0003756886
Epoch-- 995 --loss-- 0.0003731146
Epoch-- 996 --loss-- 0.00037055893
Epoch-- 997 --loss-- 0.00036801986
Epoch-- 998 --loss-- 0.00036549888
Epoch-- 999 --loss-- 0.00036299432
```

Support Vector Machine (SVM) Algorithm

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. While it can handle regression problems, SVM is particularly well-suited for classification tasks.
SVM aims to find the optimal hyperplane in an N-dimensional space to separate data points into different classes. The algorithm maximizes the margin between the closest points of different classes.

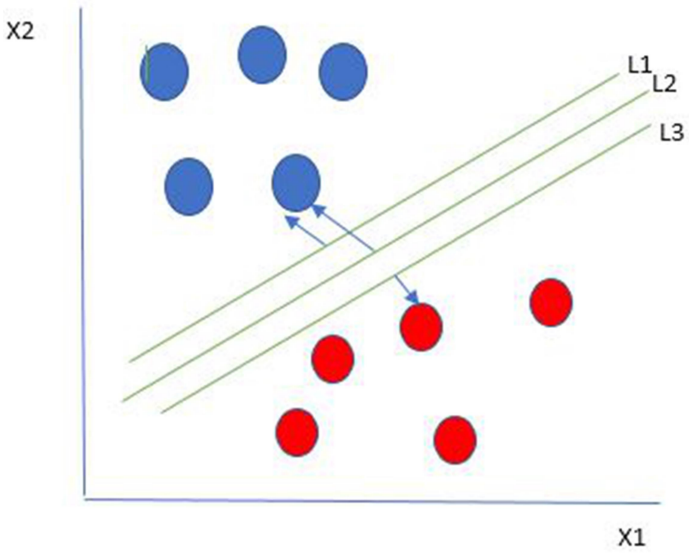
Support Vector Machine (SVM) Terminology

- **Hyperplane**: A decision boundary separating different classes in feature space, represented by the equation **wx + b = 0** in linear classification.
- **Support Vectors**: The closest data points to the hyperplane, crucial for determining the hyperplane and margin in SVM.
- **Margin**: The distance between the hyperplane and the support vectors. SVM aims to maximize this margin for better classification performance.
- **Kernel**: A function that maps data to a higher-dimensional space, enabling SVM to handle non-linearly separable data.
- **Hard Margin**: A maximum-margin hyperplane that perfectly separates the data without misclassifications.
- **Soft Margin**: Allows some misclassifications by introducing slack variables, balancing margin maximization and misclassification penalties when data is not perfectly separable.
- **C**: A regularization term balancing margin maximization and misclassification penalties. A higher C value enforces a stricter penalty for misclassifications.

- **Hinge Loss**: A loss function penalizing misclassified points or margin violations, combined with regularization in SVM.
- **Dual Problem**: Involves solving for Lagrange multipliers associated with support vectors, facilitating the kernel trick and efficient computation.

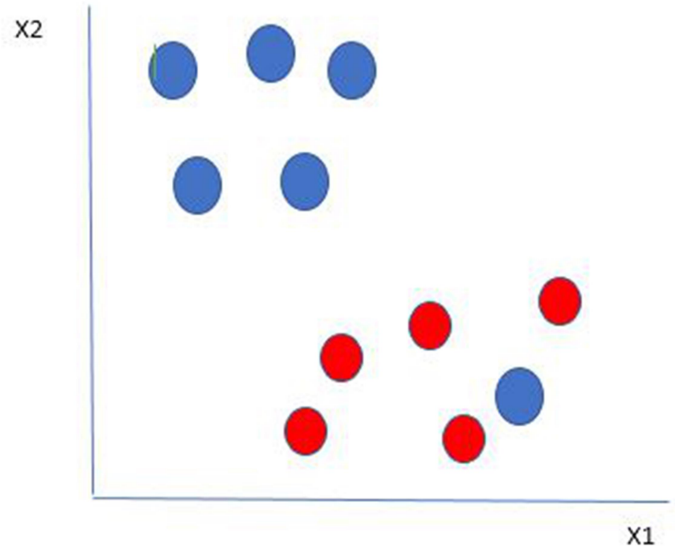
How does Support Vector Machine Algorithm Work?

The key idea behind the SVM algorithm is to find the hyperplane that best separates two classes by maximizing the margin between them. This margin is the distance from the hyperplane to the nearest data points (**support vectors**) on each side.



Multiple hyperplanes separate the data from two classes

The best hyperplane, also known as the **"hard margin,"** is the one that maximizes the distance between the hyperplane and the nearest data points from both classes. This ensures a clear separation between the classes. So, from the above figure, we choose L2 as hard margin.
Let's consider a scenario like shown below:

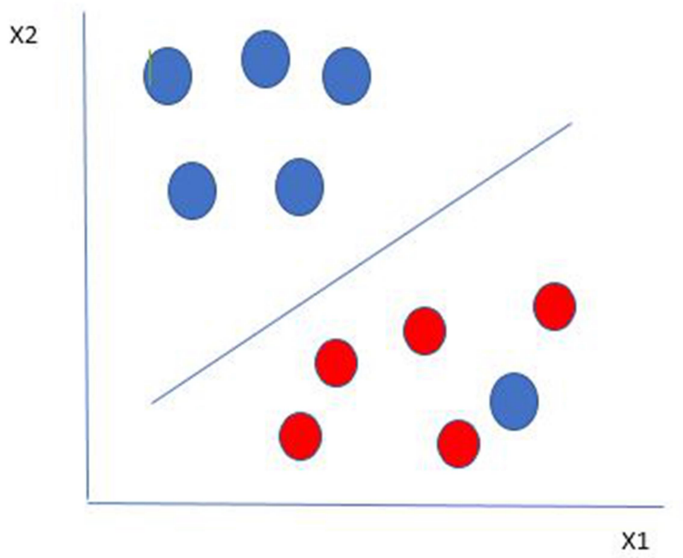


Selecting hyperplane for data with outlier

Here, we have one blue ball in the boundary of the red ball.

How does SVM classify the data?

It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.



Hyperplane which is the most optimized one

A soft margin allows for some misclassifications or violations of the margin to improve generalization. The SVM optimizes the following equation to balance margin maximization and penalty minimization:

$$\text{Objective Function} = \left(\frac{1}{\text{margin}}\right) + \lambda \sum \text{penalty}$$

Objective Function=(margin1)+λ∑penalty

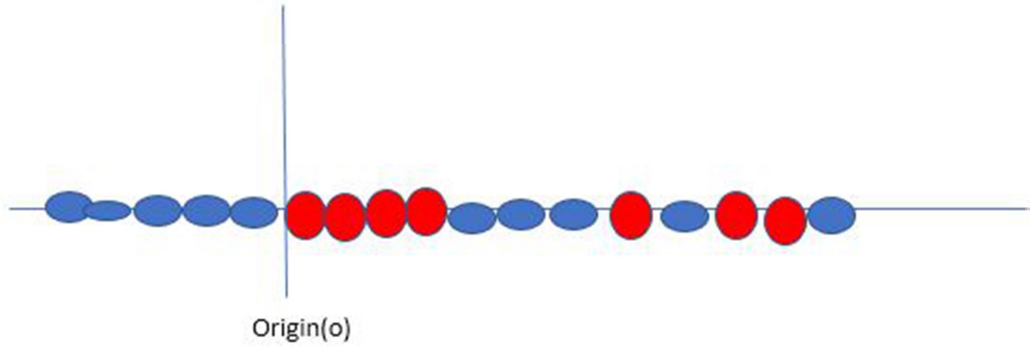
The penalty used for violations is often **hinge loss**, which has the following behavior:

- If a data point is correctly classified and within the margin, there is no penalty (loss = 0).
- If a point is incorrectly classified or violates the margin, the hinge loss increases proportionally to the distance of the violation.

Till now, we were talking about linearly separable data(the group of blue balls and red balls are separable by a straight line/linear line).

What to do if data are not linearly separable?

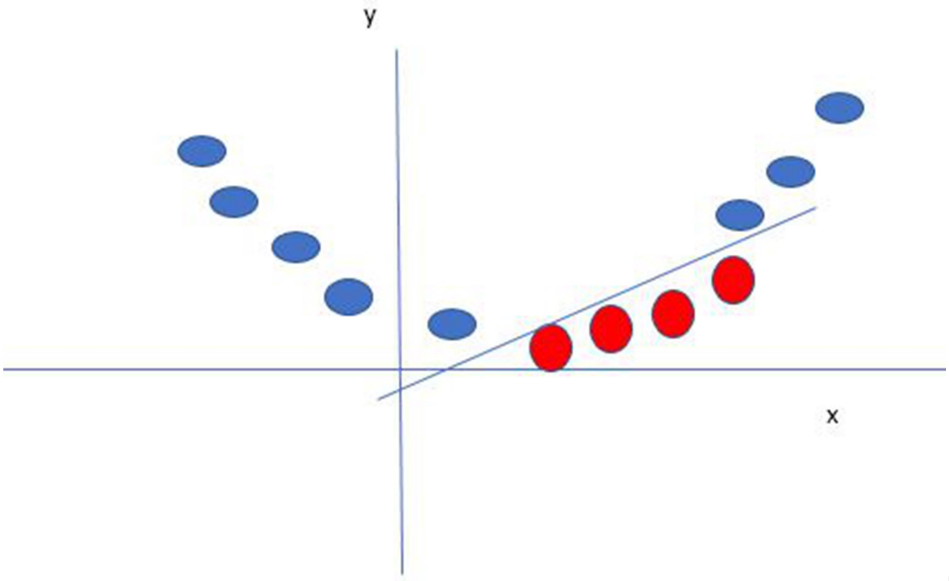
When data is not linearly separable (i.e., it can't be divided by a straight line), SVM uses a technique called **kernels** to map the data into a higher-dimensional space where it becomes separable. This transformation helps SVM find a decision boundary even for non-linear data.



Original 1D dataset for classification

A **kernel** is a function that maps data points into a higher-dimensional space without explicitly computing the coordinates in that space. This allows SVM to work efficiently with non-linear data by implicitly performing the mapping.
For example, consider data points that are not linearly separable. By applying a kernel function, SVM transforms the data points into a higher-dimensional space where they become linearly separable.

- **Linear Kernel**: For linear separability.
- **Polynomial Kernel**: Maps data into a polynomial space.
- **Radial Basis Function (RBF) Kernel**: Transforms data into a space based on distances between data points.



Mapping 1D data to 2D to become able to separate the two classes

In this case, the new variable y is created as a function of distance from the origin.

Mathematical Computation: SVM

Consider a binary classification problem with two classes, labeled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y.

The equation for the linear hyperplane can be written as:

$$wTx+b=0 w_T x+b=0$$

Where:

- ww is the normal vector to the hyperplane (the direction perpendicular to it).
- bb is the offset or bias term, representing the distance of the hyperplane from the origin along the normal vector ww.

Distance from a Data Point to the Hyperplane

The distance between a data point x_i and the decision boundary can be calculated as:

$$di=wTxi+b||w|| d_i=\frac{|w_T x_i+b|}{||w||}$$

where ||w|| represents the Euclidean norm of the weight vector w. Euclidean norm of the normal vector W

Linear SVM Classifier

Distance from a Data Point to the Hyperplane:

$$y^=\{1: wTx+b\geq 00: \ wTx+b <0 y^=\{10: w_T x+b\geq 0: \ w_T x+b <0$$

Where y^$y^$ is the predicted label of a data point.

Optimization Problem for SVM

For a linearly separable dataset, the goal is to find the hyperplane that maximizes the margin between the two classes while ensuring that all data points are correctly classified. This leads to the following optimization problem:

$$\text{minimize}_{w,b} \frac{1}{2}\|w\|^2 \quad \text{minimize}_{w,b} \frac{1}{2}\|w\|^2$$

Subject to the constraint:

$$yi(wTxi+b)\geq 1 fori=1,2,3,\cdots,m y_i(w_T x_i+b)\geq 1 for i=1,2,3,\cdots,m$$

Where:

- yiy_i is the class label (+1 or -1) for each training instance.
- xix_i is the feature vector for the ii-th training instance.
- mm is the total number of training instances.

The condition yi(wTxi+b)≥1$y_i(w^Tx_i+b)\geq 1$ ensures that each data point is correctly classified and lies outside the margin.

Soft Margin Linear SVM Classifier

In the presence of outliers or non-separable data, the SVM allows some misclassification by introducing slack variables ζiζi. The optimization problem is modified as:

minimize w,b12‖w‖2+C∑i=1mζiw,bminimize 21‖w‖2+C∑i=1mζi

Subject to the constraints:

yi(wTxi+b)≥1−ζiandζi≥0for i=1,2,…,myi(wTxi+b)≥1−ζiandζi≥0for i=1,2,…,m

Where:

- CC is a regularization parameter that controls the trade-off between margin maximization and penalty for misclassifications.
- ζiζi are slack variables that represent the degree of violation of the margin by each data point.

Dual Problem for SVM

The dual problem involves maximizing the Lagrange multipliers associated with the support vectors. This transformation allows solving the SVM optimization using kernel functions for non-linear classification.

The dual objective function is given by:

maximize α12∑i=1m∑j=1mαiαjtitjK(xi,xj)−∑i=1mαiαmaximize 21∑i=1m∑j=1mαiαjtitjK(xi,xj)−∑i=1mαi

Where:

- αiαi are the Lagrange multipliers associated with the ii-th training sample.
- titi is the class label for the iii-th training sample (+1+1+1 or −1-1−1).
- K(xi,xj)K(xi,xj) is the kernel function that computes the similarity between data points xixi and xjxj. The kernel allows SVM to handle non-linear classification problems by mapping data into a higher-dimensional space.

The dual formulation optimizes the Lagrange multipliers αiαi, and the support vectors are those training samples where αi>0αi>0.

SVM Decision Boundary

Once the dual problem is solved, the decision boundary is given by:

w=∑i=1mαitiK(xi,x)+bw=∑i=1mαitiK(xi,x)+b

Where ww is the weight vector, xx is the test data point, and bb is the bias term.

Finally, the bias term bb is determined by the support vectors, which satisfy:

ti(wTxi−b)=1⇒b=wTxi−titi(wTxi−b)=1⇒b=wTxi−ti

Where xixi is any support vector.

This completes the mathematical framework of the Support Vector Machine algorithm, which allows for both linear and non-linear classification using the dual problem and kernel trick.

Types of Support Vector Machine

Based on the nature of the decision boundary, Support Vector Machines (SVM) can be divided into two main parts:

- **Linear SVM:** Linear SVMs use a linear decision boundary to separate the data points of different classes. When the data can be precisely linearly separated, linear SVMs are very suitable. This means that a single straight line (in 2D) or a hyperplane (in higher dimensions) can entirely divide the data points into their

respective classes. A hyperplane that maximizes the margin between the classes is the decision boundary.

- **Non-Linear SVM:** Non-Linear SVM can be used to classify data when it cannot be separated into two classes by a straight line (in the case of 2D). By using kernel functions, nonlinear SVMs can handle nonlinearly separable data. The original input data is transformed by these kernel functions into a higher-dimensional feature space, where the data points can be linearly separated. A linear SVM is used to locate a nonlinear decision boundary in this modified space.

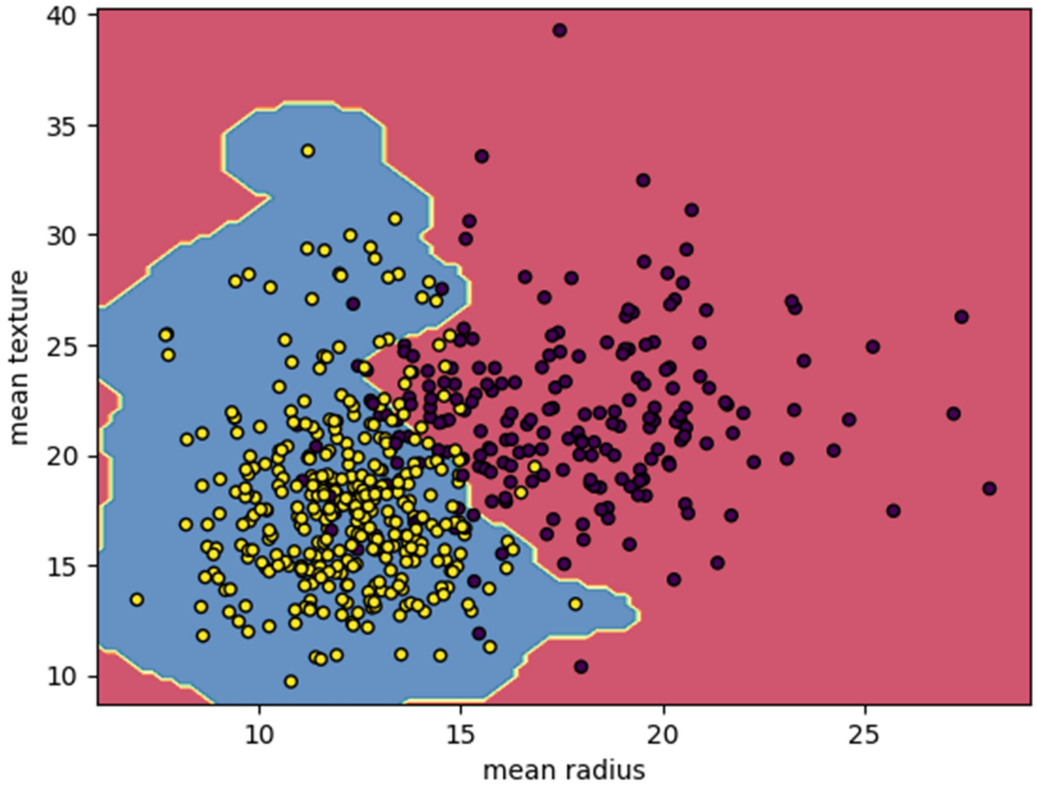
Implementing SVM Algorithm in Python

Predict if cancer is Benign or malignant. Using historical data about patients diagnosed with cancer enables doctors to differentiate malignant cases and benign ones are given independent attributes.

- Load the breast cancer dataset from sklearn.datasets
- Separate input features and target variables.
- Build and train the SVM classifiers using RBF kernel.
- Plot the scatter plot of the input features.

```
# Load the important packages
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

# Load the datasets
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

#Build the model
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
# Trained the model
svm.fit(X, y)

# Plot Decision Boundary
DecisionBoundaryDisplay.from_estimator(
        svm,
        X,
        response_method="predict",
        cmap=plt.cm.Spectral,
        alpha=0.8,
        xlabel=cancer.feature_names[0],
        ylabel=cancer.feature_names[1],
    )

# Scatter plot
plt.scatter(X[:, 0], X[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```

Output:



Breast Cancer Classifications with SVM RBF kernel

Advantages of Support Vector Machine (SVM)

1. **High-Dimensional Performance**: SVM excels in high-dimensional spaces, making it suitable for **image classification** and **gene expression analysis**.
2. **Nonlinear Capability**: Utilizing **kernel functions** like **RBF** and **polynomial**, SVM effectively handles **nonlinear relationships**.
3. **Outlier Resilience**: The **soft margin** feature allows SVM to ignore outliers, enhancing robustness in **spam detection** and **anomaly detection**.
4. **Binary and Multiclass Support**: SVM is effective for both **binary classification** and **multiclass classification**, suitable for applications in **text classification**.
5. **Memory Efficiency**: SVM focuses on **support vectors**, making it memory efficient compared to other algorithms.

Disadvantages of Support Vector Machine (SVM)

1. **Slow Training**: SVM can be slow for large datasets, affecting performance in **SVM in data mining** tasks.
2. **Parameter Tuning Difficulty**: Selecting the right **kernel** and adjusting parameters like **C** requires careful tuning, impacting **SVM algorithms**.
3. **Noise Sensitivity**: SVM struggles with noisy datasets and overlapping classes, limiting effectiveness in real-world scenarios.
4. **Limited Interpretability**: The complexity of the **hyperplane** in higher dimensions makes SVM less interpretable than other models.
5. **Feature Scaling Sensitivity**: Proper **feature scaling** is essential; otherwise, SVM models may perform poorly.

KERNEL TRICK

The kernel trick is a powerful technique in machine learning that allows linear models to be applied to nonlinear problems by implicitly mapping data into a higher-dimensional feature space. This mapping is achieved using a kernel function, which calculates the dot product between two data points in the higher-dimensional space without explicitly calculating the coordinates. The kernel trick is particularly useful in Support Vector Machines (SVMs) to handle non-linear data.

Here's a more detailed explanation:

- Implicit Mapping:

  The kernel trick avoids explicitly calculating the transformation of data into a higher-dimensional space, which can be computationally expensive. Instead, it uses a kernel function to calculate the dot product between points in the higher-dimensional space.

- Kernel Functions:

  Different kernel functions can be used, such as polynomial, radial basis function (RBF), or sigmoid kernels, each offering different nonlinear transformations.

- Linear Separability:

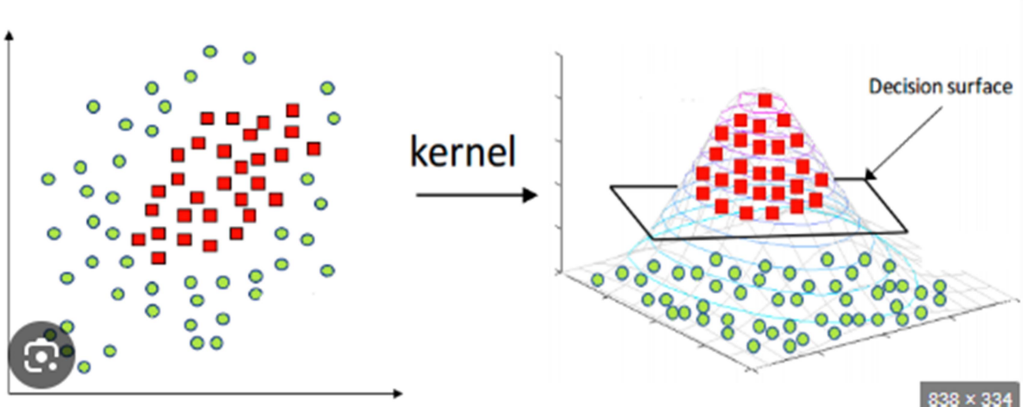
  By implicitly mapping data into a higher-dimensional space, the kernel trick enables linear classifiers to find separating hyperplanes that would not be possible in the original, lower-dimensional space.

- Computational Efficiency:

  The kernel trick provides a way to perform computations in the higher-dimensional space without needing to explicitly represent the transformed data, making it computationally efficient.

- Applications:

  The kernel trick is widely used in various machine learning applications, including classification, regression, dimensionality reduction, and clustering.



Logistic Regression

What is Logistic Regression?

Logistic regression is a **supervised machine learning algorithm** used for **classification tasks** where the goal is to predict the probability that an instance belongs to a given class or not. Logistic regression is a statistical algorithm which analyze the relationship between two data factors. The article explores the fundamentals of logistic regression, it's types and implementations.

Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1.

For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 otherwise it belongs to Class 0. It's referred to as regression because it is the extension of linear regression but is mainly used for classification problems.

Key Points:

- Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value.
- It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

Types of Logistic Regression

On the basis of the categories, Logistic Regression can be classified into three types:

1. **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
2. **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
3. **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Assumptions of Logistic Regression

We will explore the assumptions of logistic regression as understanding these assumptions is important to ensure that we are using appropriate application of the model. The assumption include:

1. Independent observations: Each observation is independent of the other. meaning there is no correlation between any input variables.
2. Binary dependent variables: It takes the assumption that the dependent variable must be binary or dichotomous, meaning it can take only two values. For more than two categories SoftMax functions are used.
3. Linearity relationship between independent variables and log odds: The relationship between the independent variables and the log odds of the dependent variable should be linear.
4. No outliers: There should be no outliers in the dataset.
5. Large sample size: The sample size is sufficiently large

Understanding Sigmoid Function

So far, we've covered the basics of logistic regression, but now let's focus on the most important function that forms the core of logistic regression**.**

- The **sigmoid function** is a mathematical function used to map the predicted values to probabilities.

- It maps any real value into another value within a range of **0 and 1**. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "**S**" form.
- The S-form curve is called the **Sigmoid function or the logistic function.**
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

How does Logistic Regression work?

The logistic regression model transforms the linear regression function continuous value output into categorical value output using a sigmoid function, which maps any real-valued set of independent variables input into a value between 0 and 1. This function is known as the logistic function.

Let the independent input features be:

$$X=[x11 \dots x1m x21 \dots x2m \vdots \ddots \vdots xn1 \dots xnm] X=\begin{bmatrix} x11 & x21 & \vdots xn1 \\ \dots\dots & \ddots & \dots x1m x2m \vdots xnm \end{bmatrix}$$

and the dependent variable is Y having only binary value i.e. 0 or 1.

$$Y=\{0 \text{ if Class1 } 1 \text{ if Class2} Y=\{01 \text{ if } Class1 \text{ if } Class2$$

then, apply the multi-linear function to the input variables X.

$$z=(\sum i=1nwixi)+bz=(\sum i=1nwixi)+b$$

Here xixi is the ith observation of X, wi=[w1,w2,w3,⋯,wm]$wi=[w1,w2,w3,\cdots,wm]$ is the weights or Coefficient, and b is the bias term also known as intercept. simply this can be represented as the dot product of weight and bias.

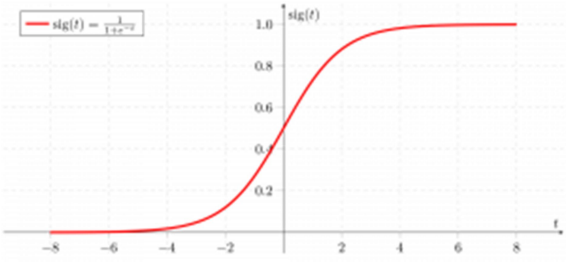
$$z=w\cdot X+bz=w\cdot X+b$$

whatever we discussed above is the linear regression.

Sigmoid Function

Now we use the sigmoid function where the input will be z and we find the probability between 0 and 1. i.e. predicted y.

$$\sigma(z)=11+e-z\sigma(z)=1+e-z1$$



Sigmoid function

As shown above, the figure sigmoid function converts the continuous variable data into the probability i.e. between 0 and 1.

- σ(z) $\sigma(z)$ tends towards 1 as z→∞$z\to\infty$
- σ(z) $\sigma(z)$ tends towards 0 as z→−∞$z\to-\infty$
- σ(z) $\sigma(z)$ is always bounded between 0 and 1

where the probability of being a class can be measured as:

$$P(y=1)=\sigma(z)P(y=0)=1-\sigma(z)P(y=1)=\sigma(z)P(y=0)=1-\sigma(z)$$

Equation of Logistic Regression:

The odd is the ratio of something occurring to something not occurring. it is different from probability as the probability is the ratio of something occurring to everything that could possibly occur. so odd will be:

$$p(x)1-p(x) =ez1-p(x)p(x) =ez$$

Applying natural log on odd. then log odd will be:

log [p(x)1−p(x)]=zlog [p(x)1−p(x)]=w·X+bp(x)1−p(x)=ew·X+b⋯Exponentiate both side sp(x)=ew·X+b·(1−p(x))p(x)=ew·X+b−ew·X+b·p(x))p(x)+ew·X+b·p(x))=ew·X+bp(x)(1+ew·

X+b)=ew·X+bp(x)=ew·X+b1+ew·X+blog[1−$p(x)$$p(x)$]log[1−$p(x)$$p(x)$]1−$p(x)$$p(x)$
$p(x)$$p(x)$$p(x)$+$ew·X+b·p(x)$)$p(x)$(1+$ew·X+b$)$p(x)$
=$z=w·X+b=ew·X+b$⋯Exponentiate both sides=$ew·X+b·(1−p(x))=ew·X+b−ew·X+b·p(x))=ew·X$
$+b=ew·X+b$=1+$ew·X+bew·X+b$

then the final logistic regression equation will be:

p(X;b,w)=ew·X+b1+ew·X+b=11+e−w·X+b$p(X;b,w)=1+ew·X+bew·X+b=1+e−w·X+b1$

Likelihood Function for Logistic Regression

The predicted probabilities will be:

- for y=1 The predicted probabilities will be: p(X;b,w) = p(x)
- for y = 0 The predicted probabilities will be: 1-p(X;b,w) = 1-p(x)

L(b,w)=∏i=1np(xi)yi(1−p(xi))1−yi$L(b,w)=∏i=1np(xi)yi(1−p(xi))1−yi$

Taking natural logs on both sides

log (L(b,w))=∑i=1nyilog p(xi)+(1−yi)log (1−p(xi))=∑i=1nyilog p(xi)+log (1−p(xi))
−yilog (1−p(xi))=∑i=1nlog (1−p(xi))+∑i=1nyilog p(xi)1−p(xi=∑i=1n−log 1−e−(w·xi
+b)+∑i=1nyi(w·xi+b)=∑i=1n−log 1+ew·xi+b+∑i=1nyi(w·xi+b)log($L(b,w)$)=$i=1∑nyi$
log$p(xi)$+(1−yi)log(1−$p(xi)$)=$i=1∑nyi$log$p(xi)$+log(1−$p(xi)$)−yilog(1−$p(xi)$)=$i=1∑n$log(1−$p(xi$
))+$i=1∑nyi$log1−$p(xip(xi)=i=1∑n$−log1−$e−(w·xi+b)+i=1∑nyi(w·xi+b)=i=1∑n$−log1+$ew·xi$
$+b+i=1∑nyi(w·xi+b)$

Gradient of the log-likelihood function

To find the maximum likelihood estimates, we differentiate w.r.t w,

∂J(l(b,w)∂wj=−∑i=nn11+ew·xi+bew·xi+bxij+∑i=1nyixij=−∑i=nnp(xi;b,w)xij+∑i=1nyixij=
∑i=nn(yi−p(xi;b,w))xij$∂wj∂J(l(b,w)=−i=n∑n1+ew·xi+b1ew·xi+bxij+i=1∑nyixij=−i=n∑np(xi$
$;b,w)xij+i=1∑nyixij=i=n∑n(yi−p(xi;b,w))xij$

Terminologies involved in Logistic Regression

Here are some common terms involved in logistic regression:

- **Independent variables:** The input characteristics or predictor factors applied to the dependent variable's predictions.
- **Dependent variable:** The target variable in a logistic regression model, which we are trying to predict.
- **Logistic function:** The formula used to represent how the independent and dependent variables relate to one another. The logistic function transforms the input variables into a probability value between 0 and 1, which represents the likelihood of the dependent variable being 1 or 0.
- **Odds:** It is the ratio of something occurring to something not occurring. it is different from probability as the probability is the ratio of something occurring to everything that could possibly occur.
- **Log-odds:** The log-odds, also known as the logit function, is the natural logarithm of the odds. In logistic regression, the log odds of the dependent variable are modeled as a linear combination of the independent variables and the intercept.
- **Coefficient:** The logistic regression model's estimated parameters, show how the independent and dependent variables relate to one another.
- **Intercept:** A constant term in the logistic regression model, which represents the log odds when all independent variables are equal to zero.
- **Maximum likelihood estimation:** The method used to estimate the coefficients of the logistic regression model, which maximizes the likelihood of observing the data given the model

Code Implementation for Logistic Regression

So far, we've covered the basics of logistic regression with all the theoritical concepts, but now let's focus on the hands on code implementation part which makes you understand the

logistic regression more clearly. We will dicuss **Binomial Logistic regression** and **Multinomial Logistic Regression** one by one.

Binomial Logistic regression:

Target variable can have only 2 possible types: "0" or "1" which may represent "**win**" vs "**loss**", "**pass**" vs "**fail**", "**dead**" vs "**alive**", etc., in this case, sigmoid functions are used, which is already discussed above.

Importing necessary libraries based on the requirement of model. This Python code shows how to use the **breast cancer dataset** to implement a Logistic Regression model for classification.

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#load the following dataset
X, y = load_breast_cancer(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=23)

clf = LogisticRegression(max_iter=10000, random_state=0)
clf.fit(X_train, y_train)

acc = accuracy_score(y_test, clf.predict(X_test)) * 100
print(f'Logistic Regression model accuracy: {acc:.2f}%')
```

Output:

```
Logistic Regression model accuracy (in %): 96.49%
```

This code loads the **breast cancer dataset** from **scikit-learn**, splits it into training and testing sets, and then trains a Logistic Regression model on the training data. The model is used to predict the labels for the test data, and the accuracy of these predictions is calculated by comparing the predicted values with the actual labels from the test set. Finally, the accuracy is printed as a percentage.

Multinomial Logistic Regression:

Target variable can have 3 or more possible types which are not ordered (i.e. types have no quantitative significance) like "**disease A**" vs "**disease B**" vs "**disease C**".

In this case, the softmax function is used in place of the sigmoid function. Softmax function for K classes will be:

softmax(zi)=ezi∑j=1Kezjsoftmax(*zi*)=∑*j*=1*Kezjezi*

Here, **K** represents the number of elements in the vector z, and i, j iterates over all the elements in the vector.

Then the probability for class c will be:

P(Y=c|X→=x)=ewc·x+bc∑k=1Kewk·x+bk*P*(*Y*=*c*|*X*=*x*)=∑*k*=1*Kewk*·*x*+*bkewc*·*x*+*bc*

In Multinomial Logistic Regression, the output variable can have **more than two possible discrete outputs**. Consider the Digit Dataset.

```
from sklearn.model_selection import train_test_split
from sklearn import datasets, linear_model, metrics

digits = datasets.load_digits()

X = digits.data
y = digits.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)

reg = linear_model.LogisticRegression(max_iter=10000, random_state=0)
reg.fit(X_train, y_train)

y_pred = reg.predict(X_test)

print(f"Logistic Regression model accuracy: {metrics.accuracy_score(y_test, y_pred) *
100:.2f}%")
```

Output:

```
Logistic Regression model accuracy(in %): 96.66%
```

How to Evaluate Logistic Regression Model?

So far, we've covered the implementation of logistic regression. Now, let's dive into the evaluation of logistic regression and understand why it's important

Evaluating the model helps us assess the model's performance and ensure it generalizes well to new data

We can evaluate the logistic regression model using the following metrics:

- **Accuracy:** Accuracy provides the proportion of correctly classified instances. Accuracy=TruePositives+TrueNegativesTotal*Accuracy=TotalTruePositives+TrueNegatives*
- **Precision:** Precision focuses on the accuracy of positive predictions. Precision=TruePositivesTruePositives+FalsePositives*Precision=TruePositives+FalsePositivesTruePositives*
- **Recall (Sensitivity or True Positive Rate):** Recall measures the proportion of correctly predicted positive instances among all actual positive instances. Recall=TruePositivesTruePositives+FalseNegatives*Recall=TruePositives+FalseNegativesTruePositives*
- **F1 Score:** F1 score is the harmonic mean of precision and recall. F1Score=2∗Precision∗RecallPrecision+Recall*F*1*Score*=2∗*Precision*+*RecallPrecision*∗*Recall*
- **Area Under the Receiver Operating Characteristic Curve (AUC-ROC):** The ROC curve plots the true positive rate against the false positive rate at various thresholds. AUC-ROC measures the area under this curve, providing an aggregate measure of a model's performance across different classification thresholds.
- **Area Under the Precision-Recall Curve (AUC-PR):** Similar to AUC-ROC, AUC-PR measures the area under the precision-recall curve, providing a summary of a model's performance across different precision-recall trade-offs.

Differences Between Linear and Logistic Regression

Now lets dive into the key differences of Linear Regression and Logistic Regression and evaluate that how they are different from each other.

The difference between linear regression and logistic regression is that linear regression output is the continuous value that can be anything while logistic regression predicts the probability that an instance belongs to a given class or not.

Linear Regression	Logistic Regression
Linear regression is used to predict the continuous dependent	Logistic regression is used to predict the categorical dependent

Linear Regression	Logistic Regression
variable using a given set of independent variables.	variable using a given set of independent variables.
Linear regression is used for solving regression problem.	It is used for solving classification problems.
In this we predict the value of continuous variables	In this we predict values of categorical variables
In this we find best fit line.	In this we find S-Curve.
Least square estimation method is used for estimation of accuracy.	Maximum likelihood estimation method is used for Estimation of accuracy.
The output must be continuous value, such as price, age, etc.	Output must be categorical value such as 0 or 1, Yes or no, etc.
It required linear relationship between dependent and independent variables.	It not required linear relationship.
There may be collinearity between the independent variables.	There should be little to no collinearity between independent variables.

Linear Regression in Machine learning

Linear regression is a type of supervised machine-learning algorithm that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets. It assumes that there is a linear relationship between the input and output, meaning the output changes at a constant rate as the input changes. This relationship is represented by a straight line.

For example We want to predict a student's exam score based on how many hours they studied. We observe that as students study more hours, their scores go up. In the example of predicting exam scores based on hours studied. Here

- Independent variable (input): Hours studied because it's the factor we control or observe.
- Dependent variable (output): Exam score because it depends on how many hours were studied.

We use the independent variable to predict the dependent variable.

Why Linear Regression is Important?

Here’s why linear regression is important:

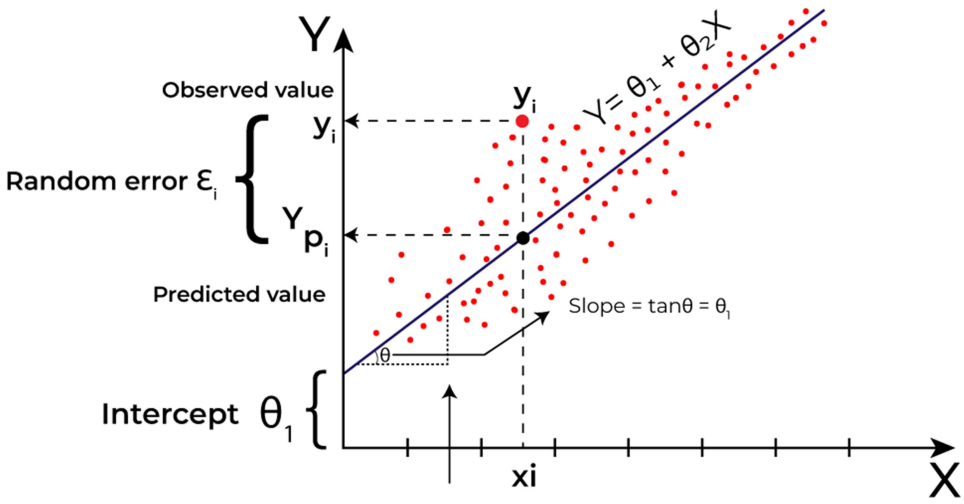
- Simplicity and Interpretability: It's easy to understand and interpret, making it a starting point for learning about machine learning.
- Predictive Ability: Helps predict future outcomes based on past data, making it useful in various fields like finance, healthcare and marketing.
- Basis for Other Models: Many advanced algorithms, like logistic regression or neural networks, build on the concepts of linear regression.
- Efficiency: It's computationally efficient and works well for problems with a linear relationship.
- Widely Used: It's one of the most widely used techniques in both statistics and machine learning for regression tasks.
- Analysis: It provides insights into relationships between variables (e.g., how much one variable influences another).

Best Fit Line in Linear Regression

In linear regression, the best-fit line is the straight line that most accurately represents the relationship between the independent variable (input) and the dependent variable (output). It is the line that minimizes the difference between the actual data points and the predicted values from the model.

1. Goal of the Best-Fit Line

The goal of linear regression is to find a straight line that minimizes the error (the difference) between the observed data points and the predicted values. This line helps us predict the dependent variable for new, unseen data.



Linear Regression

Here Y is called a dependent or target variable and X is called an independent variable also known as the predictor of Y. There are many types of functions or modules that can be used for regression. A linear function is the simplest type of function. Here, X may be a single feature or multiple features representing the problem.

2. Equation of the Best-Fit Line

For simple linear regression (with one independent variable), the best-fit line is represented by the equation

$y=mx+by=mx+b$

Where:

- y is the predicted value (dependent variable)
- x is the input (independent variable)
- m is the slope of the line (how much y changes when x changes)
- b is the intercept (the value of y when x = 0)

The best-fit line will be the one that optimizes the values of m (slope) and b (intercept) so that the predicted y values are as close as possible to the actual data points.

3. Minimizing the Error: The Least Squares Method

To find the best-fit line, we use a method called Least Squares. The idea behind this method is to minimize the sum of squared differences between the actual values (data points) and the predicted values from the line. These differences are called residuals.

The formula for residuals is:

$Residual = y_i - \hat{y}_i$

Where:

- y_i is the actual observed value
- $\hat{y}_i$ is the predicted value from the line for that x_i

The least squares method minimizes the sum of the squared residuals:

$Sum of squared errors (SSE) = \Sigma (y_i - \hat{y}_i)^2$

This method ensures that the line best represents the data, where the sum of the squared differences between the predicted values and actual values is as small as possible.

4. Interpretation of the Best-Fit Line

- Slope (m): The slope of the best-fit line indicates how much the dependent variable (y) changes with each unit change in the independent variable (x). For example, if the slope is 5, it means that for every 1-unit increase in x, the value of y increases by 5 units.
- Intercept (b): The intercept represents the predicted value of y when x = 0. It's the point where the line crosses the y-axis.

In linear regression some hypothesis are made to ensure reliability of the model's results.

Limitations

- *Assumes Linearity: The method assumes the relationship between the variables is linear. If the relationship is non-linear, linear regression might not work well.*
- *Sensitivity to Outliers: Outliers can significantly affect the slope and intercept, skewing the best-fit line.*

Hypothesis function in Linear Regression

In linear regression, the hypothesis function is the equation used to make predictions about the dependent variable based on the independent variables. It represents the relationship between the input features and the target output.

For a simple case with one independent variable, the hypothesis function is:

$h(x) = \beta_0 + \beta_1 x$

Where:

- h(x)(ory^) $h(x)$($ory\hat{}$) is the predicted value of the dependent variable (y).
- x is the independent variable.
- β_0 is the intercept, representing the value of y when x is 0.
- β_1 is the slope, indicating how much y changes for each unit change in x.

For multiple linear regression (with more than one independent variable), the hypothesis function expands to:

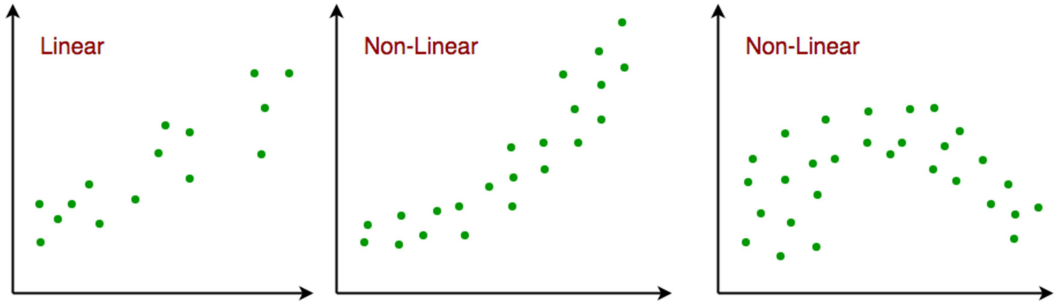
$h(x_1, x_2, \ldots, x_k) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k$

Where:

- $x_1, x_2, \ldots, x_k$ are the independent variables.
- β_0 is the intercept.
- $\beta_1, \beta_2, \ldots, \beta_k$ are the coefficients, representing the influence of each respective independent variable on the predicted output.

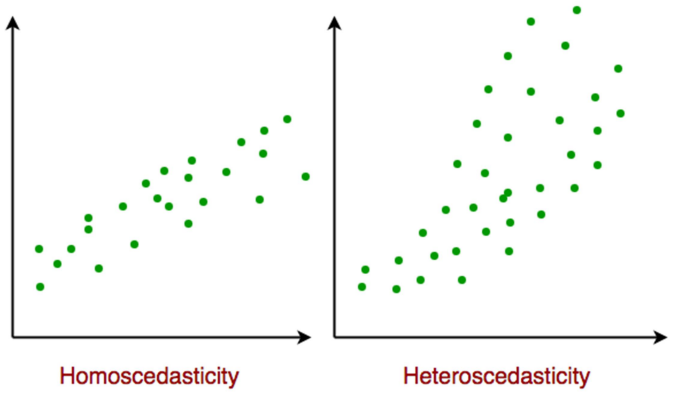
Assumptions of the Linear Regression

1. Linearity: The relationship between inputs (X) and the output (Y) is a straight line.



Linearity

2. Independence of Errors: The errors in predictions should not affect each other.
3. Constant Variance (Homoscedasticity): The errors should have equal spread across all values of the input. If the spread changes (like fans out or shrinks), it's called heteroscedasticity and it's a problem for the model.



Homoscedasticity

4. Normality of Errors: The errors should follow a normal (bell-shaped) distribution.
5. No Multicollinearity*(for multiple regression)*: Input variables shouldn't be too closely related to each other.
6. No Autocorrelation: Errors shouldn't show repeating patterns, especially in time-based data.
7. Additivity: The total effect on Y is just the sum of effects from each X, no mixing or interaction between them.'

To understand Multicollinearity in detail refer to article: Multicollinearity.

Types of Linear Regression

When there is only one independent feature it is known as Simple Linear Regression or Univariate Linear Regression and when there are more than one feature it is known as Multiple Linear Regression orMultivariate Regression.

1. Simple Linear Regression

Simple linear regression is used when we want to predict a target value (dependent variable) using only one input feature (independent variable). It assumes a straight-line relationship between the two.

Formula:

$y^=\theta 0+\theta 1xy^=\theta 0+\theta 1x$
Where:

- y^$y^$ is the predicted value
- x is the input (independent variable)
- θ0$\theta 0$ is the intercept (value of y^$y^$ when x=0)
- θ1$\theta 1$ is the slope or coefficient (how much y^$y^$ changes with one unit of x)

Example:
Predicting a person's salary (y) based on their years of experience (x).
2. Multiple Linear Regression
Multiple linear regression involves more than one independent variable and one dependent variable. The equation for multiple linear regression is:
$y^=\theta 0+\theta 1x1+\theta 2x2+\cdots+\theta nxny^=\theta 0+\theta 1x1+\theta 2x2+\cdots+\theta nxn$
where:

- y^$y^$ is the predicted value
- x1,x2,…,xnx1,$x2$,…,xn are the independent variables
- θ1,θ2,…,θnθ1,$\theta 2$,…,θn are the coefficients (weights) corresponding to each predictor.
- θ0$\theta 0$ is the intercept.

The goal of the algorithm is to find the best Fit Line equation that can predict the values based on the independent variables.
In regression set of records are present with X and Y values and these values are used to learn a function so if you want to predict Y from an unknown X this learned function can be used. In regression we have to find the value of Y, So, a function is required that predicts continuous Y in the case of regression given X as independent features.
Use Case of Multiple Linear Regression
Multiple linear regression allows us to analyze relationship between multiple independent variables and a single dependent variable. Here are some use cases:

- Real Estate Pricing: In real estate MLR is used to predict property prices based on multiple factors such as location, size, number of bedrooms, etc. This helps buyers and sellers understand market trends and set competitive prices.
- Financial Forecasting: Financial analysts use MLR to predict stock prices or economic indicators based on multiple influencing factors such as interest rates, inflation rates and market trends. This enables better investment strategies and risk management24.
- Agricultural Yield Prediction: Farmers can use MLR to estimate crop yields based on several variables like rainfall, temperature, soil quality and fertilizer usage. This information helps in planning agricultural practices for optimal productivity
- E-commerce Sales Analysis: An e-commerce company can utilize MLR to assess how various factors such as product price, marketing promotions and seasonal trends impact sales.

Now that we have understood about linear regression, its assumption and its type now we will learn how to make a linear regression model.
Cost function for Linear Regression
As we have discussed earlier about best fit line in linear regression, its not easy to get it easily in real life cases so we need to calculate errors that affects it. These errors need to be calculated to mitigate them. The difference between the predicted value Y^ $Y^$ and the true value Y and it is called cost function or the loss function.
In Linear Regression, the Mean Squared Error (MSE) cost function is employed, which calculates the average of the squared errors between the predicted values y^iy^i and the actual values yiyi. The purpose is to determine the optimal values for the intercept θ1$\theta 1$ and the coefficient of the input feature θ2$\theta 2$ providing the best-fit line for the given data points. The linear equation expressing this relationship is y^i=θ1+θ2xiy^$i=\theta 1+\theta 2xi$.

MSE function can be calculated as:

$$Cost\ function(J)=1n\sum ni(yi\hat{}-yi)2Cost\ function(J)=n1\sum ni(yi\hat{}-yi)2$$

Utilizing the MSE function, the iterative process of gradient descent is applied to update the values of \θ1&θ2θ1&θ2. This ensures that the MSE value converges to the global minima, signifying the most accurate fit of the linear regression line to the dataset.

This process involves continuously adjusting the parameters \(\theta_1\) and \(\theta_2\) based on the gradients calculated from the MSE. The final result is a linear regression line that minimizes the overall squared differences between the predicted and actual values, providing an optimal representation of the underlying relationship in the data.

Now we have calculated loss function we need to optimize model to mtigate this error and it is done through gradient descent.

Gradient Descent for Linear Regression

A linear regression model can be trained using the optimization algorithm gradient descent by iteratively modifying the model's parameters to reduce the mean squared error (MSE) of the model on a training dataset. To update θ1 and θ2 values in order to reduce the Cost function (minimizing RMSE value) and achieve the best-fit line the model uses Gradient Descent. The idea is to start with random θ1 and θ2 values and then iteratively update the values, reaching minimum cost.

A gradient is nothing but a derivative that defines the effects on outputs of the function with a little bit of variation in inputs.

Let's differentiate the cost function(J) with respect to θ1 θ1

$$J\theta1'=\partial J(\theta1,\theta2)\partial\theta1=\partial\partial\theta1[1n(\sum i=1n(y\hat{}i-yi)2)]=1n[\sum i=1n2(y\hat{}i-yi)(\partial\partial\theta1(y\hat{}i-yi))]=1n[\sum i=1n2(y\hat{}i-yi)(\partial\partial\theta1(\theta1+\theta2xi-yi))]=1n[\sum i=1n2(y\hat{}i-yi)(1+0-0)]=1n[\sum i=1n(y\hat{}i-yi)(2)]=2n\sum i=1n(y\hat{}i-yi)J\theta1'=\partial\theta1\partial J(\theta1,\theta2)=\partial\theta1\partial[n1(i=1\sum n(y\hat{}i-yi)2)]=n1[i=1\sum n2(y\hat{}i-yi)(\partial\theta1\partial(y\hat{}i-yi))]=n1[i=1\sum n2(y\hat{}i-yi)(\partial\theta1\partial(\theta1+\theta2xi-yi))]=n1[i=1\sum n2(y\hat{}i-yi)(1+0-0)]=n1[i=1\sum n(y\hat{}i-yi)(2)]=n2i=1\sum n(y\hat{}i-yi)$$

Let's differentiate the cost function(J) with respect to θ2θ2

$$J\theta2'=\partial J(\theta1,\theta2)\partial\theta2=\partial\partial\theta2[1n(\sum i=1n(y\hat{}i-yi)2)]=1n[\sum i=1n2(y\hat{}i-yi)(\partial\partial\theta2(y\hat{}i-yi))]=1n[\sum i=1n2(y\hat{}i-yi)(\partial\partial\theta2(\theta1+\theta2xi-yi))]=1n[\sum i=1n2(y\hat{}i-yi)(0+xi-0)]=1n[\sum i=1n(y\hat{}i-yi)(2xi)]=2n\sum i=1n(y\hat{}i-yi)\cdot xiJ\theta2'=\partial\theta2\partial J(\theta1,\theta2)=\partial\theta2\partial[n1(i=1\sum n(y\hat{}i-yi)2)]=n1[i=1\sum n2(y\hat{}i-yi)(\partial\theta2\partial(y\hat{}i-yi))]=n1[i=1\sum n2(y\hat{}i-yi)(\partial\theta2\partial(\theta1+\theta2xi-yi))]=n1[i=1\sum n2(y\hat{}i-yi)(0+xi-0)]=n1[i=1\sum n(y\hat{}i-yi)(2xi)]=n2i=1\sum n(y\hat{}i-yi)\cdot xi$$

Finding the coefficients of a linear equation that best fits the training data is the objective of linear regression. By moving in the direction of the Mean Squared Error negative gradient with respect to the coefficients, the coefficients can be changed. And the respective intercept and coefficient of X will be if α α is the learning rate.

Gradient Descent

$$\theta1=\theta1-\alpha(J\theta1')=\theta1-\alpha(2n\sum i=1n(y\hat{}i-yi))\theta2=\theta2-\alpha(J\theta2')=\theta2-\alpha(2n\sum i=1n(y\hat{}i-yi)\cdot xi)\theta1=\theta1-\alpha(J\theta1')=\theta1-\alpha(n2i=1\sum n(y\hat{}i-yi))\theta2=\theta2-\alpha(J\theta2')=\theta2-\alpha(n2i=1\sum n(y\hat{}i-yi)\cdot xi)$$

After optimizing our model, we evaluate our models accuracy to see how well it will perform in real world scenario.

Evaluation Metrics for Linear Regression

A variety of evaluation measures can be used to determine the strength of any linear regression model. These assessment metrics often give an indication of how well the model is producing the observed outputs.

The most common measurements are:

1. Mean Square Error (MSE)

Mean Squared Error (MSE) is an evaluation metric that calculates the average of the squared differences between the actual and predicted values for all the data points. The difference is squared to ensure that negative and positive differences don't cancel each other out.

$$MSE=1n\sum i=1n(yi-yi\hat{})2MSE=n1\sum i=1n(yi-yi)2$$

Here,

- n is the number of data points.
- yi is the actual or observed value for the ith data point.
- yi^*yi* is the predicted value for the ith data point.

MSE is a way to quantify the accuracy of a model's predictions. MSE is sensitive to outliers as large errors contribute significantly to the overall score.

2. Mean Absolute Error (MAE)

Mean Absolute Error is an evaluation metric used to calculate the accuracy of a regression model. MAE measures the average absolute difference between the predicted values and actual values.

Mathematically, MAE is expressed as:

$$MAE=1n\sum i=1n|Yi-Yi\hat{}|MAE=n1\sum i=1n|Yi-Yi|$$

Here,

- n is the number of observations
- Yi represents the actual values.
- Yi^*Yi* represents the predicted values

Lower MAE value indicates better model performance. It is not sensitive to the outliers as we consider absolute differences.

3. Root Mean Squared Error (RMSE)

The square root of the residuals' variance is the Root Mean Squared Error. It describes how well the observed data points match the expected values, or the model's absolute fit to the data.

In mathematical notation, it can be expressed as:

$$RMSE=RSSn=\sum i=2n(yiactual-yipredicted)2nRMSE=nRSS=n\sum i=2n(yiactual-yipredicted)2$$

Rather than dividing the entire number of data points in the model by the number of degrees of freedom, one must divide the sum of the squared residuals to obtain an unbiased estimate. Then, this figure is referred to as the Residual Standard Error (RSE).

In mathematical notation, it can be expressed as:

$$RMSE=RSSn=\sum i=2n(yiactual-yipredicted)2(n-2)RMSE=nRSS=(n-2)\sum i=2n(yiactual-yipredicted)2$$

RSME is not as good of a metric as R-squared. Root Mean Squared Error can fluctuate when the units of the variables vary since its value is dependent on the variables' units (it is not a normalized measure).

4. Coefficient of Determination (R-squared)

R-Squared is a statistic that indicates how much variation the developed model can explain or capture. It is always in the range of 0 to 1. In general, the better the model matches the data, the greater the R-squared number.

In mathematical notation, it can be expressed as:

$$R2=1-(RSSTSS)R2=1-(TSSRSS)$$

- Residual sum of Squares (RSS): The sum of squares of the residual for each data point in the plot or data is known as the residual sum of squares, or RSS. It is a measurement of the difference between the output that was observed and what was anticipated.

$$RSS=\sum i=1n(yi-b0-b1xi)2RSS=\sum i=1n(yi-b0-b1xi)2$$

- Total Sum of Squares (TSS): The sum of the data points' errors from the answer variable's mean is known as the total sum of squares, or TSS.

$$TSS=\sum i=1n(y-yi\bar{})2TSS=\sum i=1n(y-yi)2.$$

R squared metric is a measure of the proportion of variance in the dependent variable that is explained the independent variables in the model.

5. Adjusted R-Squared Error

Adjusted R2 measures the proportion of variance in the dependent variable that is explained by independent variables in a regression model. Adjusted R-square accounts the number of predictors in the model and penalizes the model for including irrelevant predictors that don't contribute significantly to explain the variance in the dependent variables.

Mathematically, adjusted R2 is expressed as:

$$AdjustedR2=1−((1−R2).(n−1)n−k−1)AdjustedR2=1−(n−k−1(1−R2).(n−1))$$

Here,

- n is the number of observations
- k is the number of predictors in the model
- R2 is coeeficient of determination

Adjusted R-square helps to prevent overfitting. It penalizes the model with additional predictors that do not contribute significantly to explain the variance in the dependent variable.

While evaluation metrics help us measure the performance of a model, regularization helps in improving that performance by addressing overfitting and enhancing generalization.

Regularization Techniques for Linear Models

1. Lasso Regression (L1 Regularization)

Lasso Regression is a technique used for regularizing a linear regression model, it adds a penalty term to the linear regression objective function to prevent overfitting.

The objective function after applying lasso regression is:

$$J(θ)=12m∑i=1m(yi^−yi)2+λ∑j=1n|θj|J(θ)=2m1∑i=1m(yi−yi)2+λ∑j=1n|θj|$$

- the first term is the least squares loss, representing the squared difference between predicted and actual values.
- the second term is the L1 regularization term, it penalizes the sum of absolute values of the regression coefficient θj.

2. Ridge Regression (L2 Regularization)

Ridge regression is a linear regression technique that adds a regularization term to the standard linear objective. Again, the goal is to prevent overfitting by penalizing large coefficient in linear regression equation. It useful when the dataset has multicollinearity where predictor variables are highly correlated.

The objective function after applying ridge regression is:

$$J(θ)=12m∑i=1m(yi^−yi)2+λ∑j=1nθj2J(θ)=2m1∑i=1m(yi−yi)2+λ∑j=1nθj2$$

- the first term is the least squares loss, representing the squared difference between predicted and actual values.
- the second term is the L1 regularization term, it penalizes the sum of square of values of the regression coefficient θj.

3. Elastic Net Regression

Elastic Net Regression is a hybrid regularization technique that combines the power of both L1 and L2 regularization in linear regression objective.

$$J(θ)=12m∑i=1m(yi^−yi)2+α\lambda∑j=1n|θj|+12(1−α)λ∑j=1nθj2J(θ)=2m1∑i=1m(yi−yi)2+α\lambda∑j=1n|θj|+21(1−α)λ∑j=1nθj2$$

- the first term is least square loss.
- the second term is L1 regularization and third is ridge regression.
- λ is the overall regularization strength.
- α controls the mix between L1 and L2 regularization.

The linear regression line provides valuable insights into the relationship between the two variables. It represents the best-fitting line that captures the overall trend of how a dependent variable (Y) changes in response to variations in an independent variable (X).

- Positive Linear Regression Line: A positive linear regression line indicates a direct relationship between the independent variable (X) and the dependent variable (Y). This means that as the value of X increases, the value of Y also increases. The slope of a positive linear regression line is positive, meaning that the line slants upward from left to right.
- Negative Linear Regression Line: A negative linear regression line indicates an inverse relationship between the independent variable (X) and the dependent variable (Y). This means that as the value of X increases, the value of Y decreases. The slope of a negative linear regression line is negative, meaning that the line slants downward from left to right.

Applications of Linear Regression

Linear regression is used in many different fields including finance, economics and psychology to understand and predict the behavior of a particular variable.

For example linear regression is widely used in finance to analyze relationships and make predictions. It can model how a company's earnings per share (EPS) influence its stock price. If the model shows that a $1 increase in EPS results in a $15 rise in stock price, investors gain insights into the company's valuation. Similarly, linear regression can forecast currency values by analyzing historical exchange rates and economic indicators, helping financial professionals make informed decisions and manage risks effectively.

Also read - Linear Regression - In Simple Words, with real-life Examples

Advantages and Disadvantages of Linear regression

Advantages of Linear Regression

- Linear regression is a relatively simple algorithm, making it easy to understand and implement. The coefficients of the linear regression model can be interpreted as the change in the dependent variable for a one-unit change in the independent variable, providing insights into the relationships between variables.
- Linear regression is computationally efficient and can handle large datasets effectively. It can be trained quickly on large datasets, making it suitable for real-time applications.
- Linear regression is relatively robust to outliers compared to other machine learning algorithms. Outliers may have a smaller impact on the overall model performance.
- Linear regression often serves as a good baseline model for comparison with more complex machine learning algorithms.
- Linear regression is a well-established algorithm with a rich history and is widely available in various machine learning libraries and software packages.

Disadvantages of Linear Regression

- Linear regression assumes a linear relationship between the dependent and independent variables. If the relationship is not linear, the model may not perform well.
- Linear regression is sensitive to multicollinearity, which occurs when there is a high correlation between independent variables. Multicollinearity can inflate the variance of the coefficients and lead to unstable model predictions.
- Linear regression assumes that the features are already in a suitable form for the model. Feature engineering may be required to transform features into a format that can be effectively used by the model.
- Linear regression is susceptible to both overfitting and underfitting. Overfitting occurs when the model learns the training data too well and fails to generalize to unseen data. Underfitting occurs when the model is too simple to capture the underlying relationships in the data.

- Linear regression provides limited explanatory power for complex relationships between variables. More advanced machine learning techniques may be necessary for deeper insights.

Multi-Layer Perceptron Learning in Tensorflow

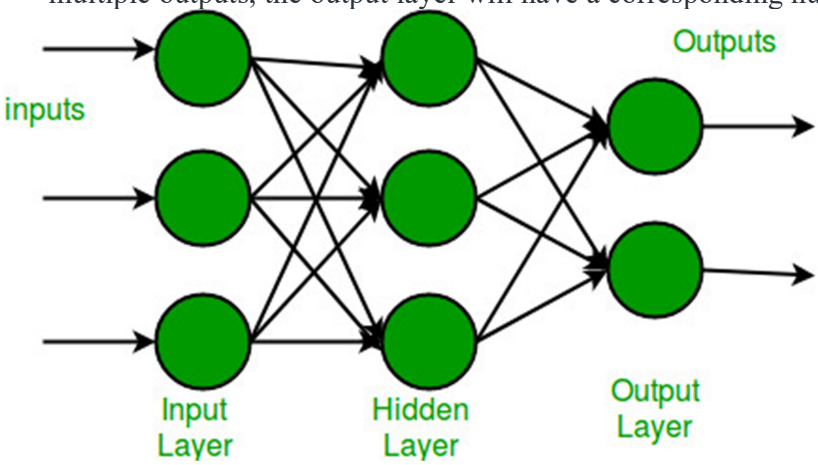
Multi-Layer Perceptron (MLP) is an artificial neural network widely used for solving classification and regression tasks.

MLP consists of fully connected dense layers that transform input data from one dimension to another. It is called *"multi-layer"* because it contains an input layer, one or more hidden layers, and an output layer. The purpose of an MLP is to model complex relationships between inputs and outputs, making it a powerful tool for various machine learning tasks.

Pre-requisites: *Neural Network*, *Artificial Neural Network*, *Perceptron*

Key Components of Multi-Layer Perceptron (MLP)

- **Input Layer**: Each neuron (or node) in this layer corresponds to an input feature. For instance, if you have three input features, the input layer will have three neurons.
- **Hidden Layers**: An MLP can have any number of hidden layers, with each layer containing any number of nodes. These layers process the information received from the input layer.
- **Output Layer**: The output layer generates the final prediction or result. If there are multiple outputs, the output layer will have a corresponding number of neurons.



Every connection in the diagram is a representation of the fully connected nature of an MLP. This means that every node in one layer connects to every node in the next layer. As the data moves through the network, each layer transforms it until the final output is generated in the output layer.

Working of Multi-Layer Perceptron

Let's delve in to the working of the multi-layer perceptron. The key mechanisms such as forward propagation, loss function, backpropagation, and optimization.

Step 1: Forward Propagation

In **forward propagation**, the data flows from the input layer to the output layer, passing through any hidden layers. Each neuron in the hidden layers processes the input as follows:

1. **Weighted Sum**: The neuron computes the weighted sum of the inputs:
   - $z=\sum iwixi+bz=\sum iwixi+b$
     - Where:

    - xix_i is the input feature.
    - wiw_i is the corresponding weight.
    - bb is the bias term.
2. **Activation Function**: The weighted sum z is passed through an activation function to introduce non-linearity. Common activation functions include:
    - **Sigmoid**: σ(z)=11+e−zσ(z)=1+e−z1
    - **ReLU (Rectified Linear Unit)**: f(z)=max (0,z)f(z)=max(0,z)
    - **Tanh (Hyperbolic Tangent)**: tanh (z)=21+e−2z−1tanh(z)=1+e−2z2−1

Step 2: Loss Function

Once the network generates an output, the next step is to calculate the loss using a loss function. In supervised learning, this compares the predicted output to the actual label. For a classification problem, the commonly used **binary cross-entropy** loss function is:

L=−1N∑i=1N[yilog (y^i)+(1−yi)log (1−y^i)]L=−N1∑i=1N[yilog(y^i)+(1−yi)log(1−y^i)]

Where:

- yiyi is the actual label.
- y^iy^i is the predicted label.
- NN is the number of samples.

For regression problems, the **mean squared error (MSE)** is often used:

MSE=1N∑i=1N(yi−y^i)2MSE=N1∑i=1N(yi−y^i)2

Step 3: Backpropagation

The goal of training an MLP is to minimize the loss function by adjusting the network's weights and biases. This is achieved through **backpropagation**:

1. **Gradient Calculation**: The gradients of the loss function with respect to each weight and bias are calculated using the chain rule of calculus.
2. **Error Propagation**: The error is propagated back through the network, layer by layer.
3. **Gradient Descent**: The network updates the weights and biases by moving in the opposite direction of the gradient to reduce the loss: w=w−η⋅∂L∂ww=w−η⋅∂w∂L
    - Where:
        - ww is the weight.
        - ηη is the learning rate.
        - ∂L∂w∂w∂L is the gradient of the loss function with respect to the weight.

Step 4: Optimization

MLPs rely on optimization algorithms to iteratively refine the weights and biases during training. Popular optimization methods include:

- **Stochastic Gradient Descent (SGD)**: Updates the weights based on a single sample or a small batch of data: w=w−η⋅∂L∂ww=w−η⋅∂w∂L
- **Adam Optimizer**: An extension of SGD that incorporates momentum and adaptive learning rates for more efficient training:
    - mt=β1mt−1+(1−β1)⋅gtmt=β1mt−1+(1−β1)⋅gt
    - vt=β2vt−1+(1−β2)⋅gt2vt=β2vt−1+(1−β2)⋅gt2
- Here, gtgt represents the gradient at time tt, and β1,β2β1,β2 are decay rates.

Now that we are done with the theory part of multi-layer perception, let's go ahead and implement some code in python using the TensorFlow library.

Advantages of Multi Layer Perceptron

- **Versatility**: MLPs can be applied to a variety of problems, both classification and regression.
- **Non-linearity**: Thanks to activation functions, MLPs can model complex, non-linear relationships in data.

- **Parallel Computation**: With the help of GPUs, MLPs can be trained quickly by taking advantage of parallel computing.

Disadvantages of Multi Layer Perceptron

- **Computationally Expensive**: MLPs can be slow to train, especially on large datasets with many layers.
- **Prone to Overfitting**: Without proper regularization techniques, MLPs can overfit the training data, leading to poor generalization.
- **Sensitivity to Data Scaling**: MLPs require properly normalized or scaled data for optimal performance.

The **Multilayer Perceptron** has the ability to learn complex patterns from data makes it a valuable tool in machine learning. Whether you're working with structured data, images, or text, understanding how MLP works can open doors to solving a wide range of problems.

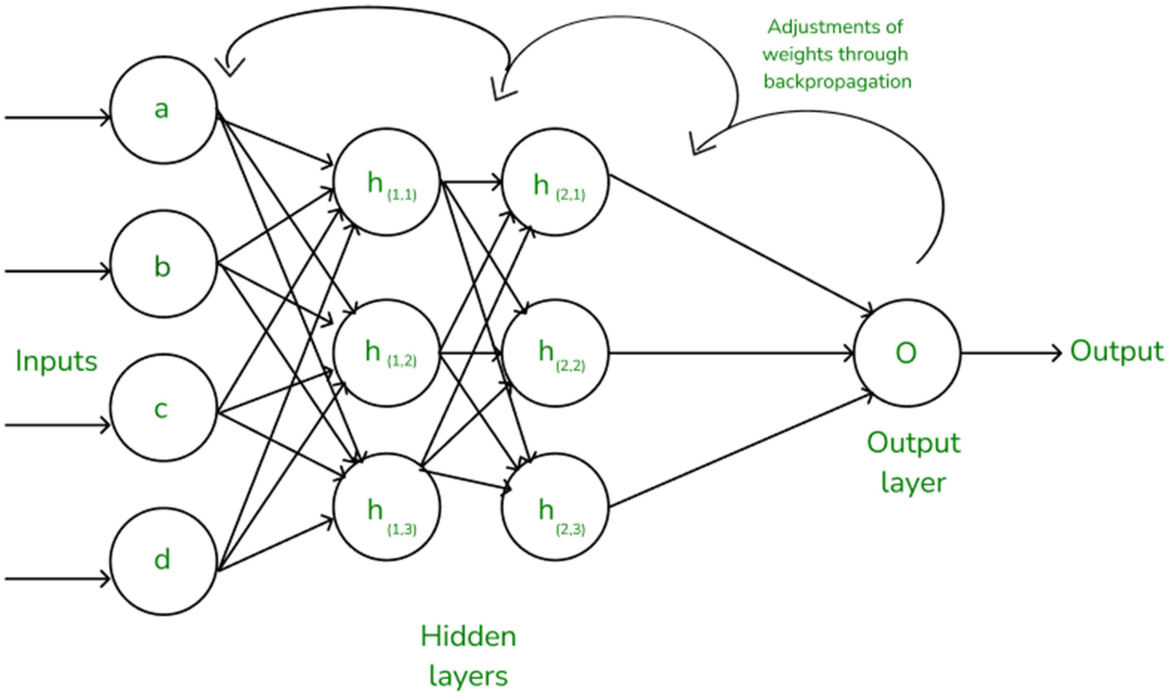
Backpropagation in Neural Network

Backpropagation is also known as "*Backward Propagation of Errors*" and it is a method used to train **neural network** . Its goal is to reduce the difference between the model's predicted output and the actual output by adjusting the weights and biases in the network. In this article we will explore what backpropagation is, why it is crucial in machine learning and how it works.

What is Backpropagation?

Backpropagation is a technique used in deep learning to train artificial neural networks particularly **feed-forward networks**. It works iteratively to adjust weights and bias to minimize the cost function.

In each epoch the model adapts these parameters reducing loss by following the error gradient. Backpropagation often uses optimization algorithms like **gradient descent** or **stochastic gradient descent**. The algorithm computes the gradient using the chain rule from calculus allowing it to effectively navigate complex layers in the neural network to minimize the cost function.



Fig(a) A simple illustration of how the backpropagation works by adjustments of weights

Backpropagation plays a critical role in how neural networks improve over time. Here's why:

1. **Efficient Weight Update**: It computes the gradient of the loss function with respect to each weight using the chain rule making it possible to update weights efficiently.
2. **Scalability**: The backpropagation algorithm scales well to networks with multiple layers and complex architectures making deep learning feasible.
3. **Automated Learning**: With backpropagation the learning process becomes automated and the model can adjust itself to optimize its performance.

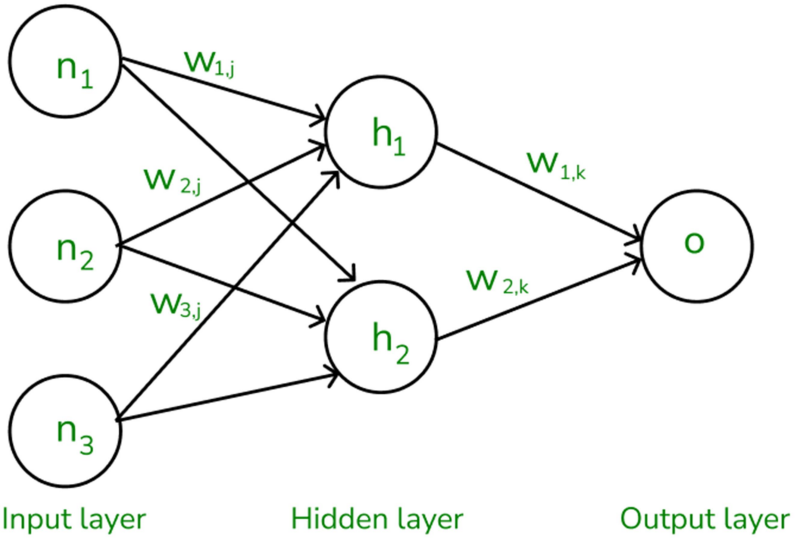
Working of Backpropagation Algorithm

The Backpropagation algorithm involves two main steps: the **Forward Pass** and the **Backward Pass**.

How Does Forward Pass Work?

In **forward pass** the input data is fed into the input layer. These inputs combined with their respective weights are passed to hidden layers. For example in a network with two hidden layers (h1 and h2 as shown in Fig. (a)) the output from h1 serves as the input to h2. Before applying an activation function, a bias is added to the weighted inputs.

Each hidden layer computes the weighted sum (`a`) of the inputs then applies an activation function like **ReLU (Rectified Linear Unit)** to obtain the output (`o`). The output is passed to the next layer where an activation function such as **softmax** converts the weighted outputs into probabilities for classification.



The forward pass using weights and biases

How Does the Backward Pass Work?

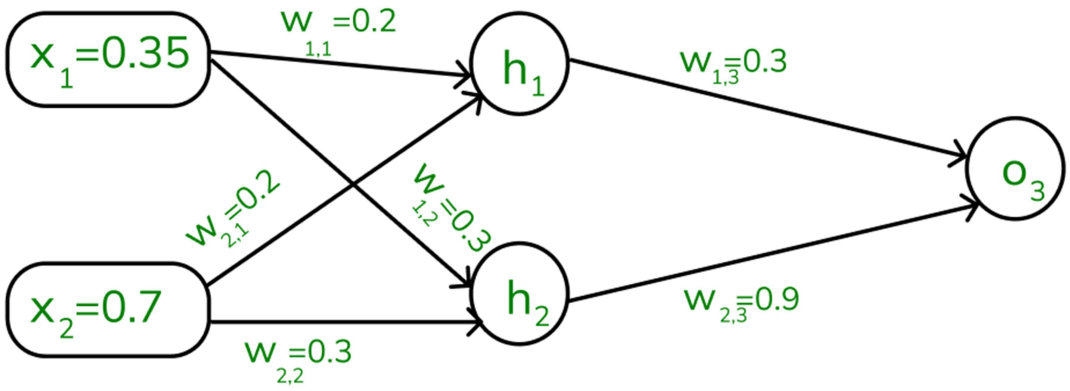
In the backward pass the error (the difference between the predicted and actual output) is propagated back through the network to adjust the weights and biases. One common method for error calculation is the **Mean Squared Error (MSE)** given by:

MSE=(Predicted Output−Actual Output)2*MSE*=(Predicted Output−Actual Output)2

Once the error is calculated the network adjusts weights using **gradients** which are computed with the chain rule. These gradients indicate how much each weight and bias should be adjusted to minimize the error in the next iteration. The backward pass continues layer by layer ensuring that the network learns and improves its performance. The activation function through its derivative plays a crucial role in computing these gradients during backpropagation.

Example of Backpropagation in Machine Learning

Let's walk through an example of backpropagation in machine learning. Assume the neurons use the sigmoid activation function for the forward and backward pass. The target output is 0.5, and the learning rate is 1.



Example (1) of backpropagation sum

Forward Propagation

1. Initial Calculation

The weighted sum at each node is calculated using:

$$aj=\sum(wi,j*xi)aj=\sum(wi,j*xi)$$

Where,

- aj*aj* is the weighted sum of all the inputs and weights at each node
- wi,j*wi,j* represents the weights between the ith*ith*input and the jth*jth* neuron
- xi*xi* represents the value of the ith*ith* input

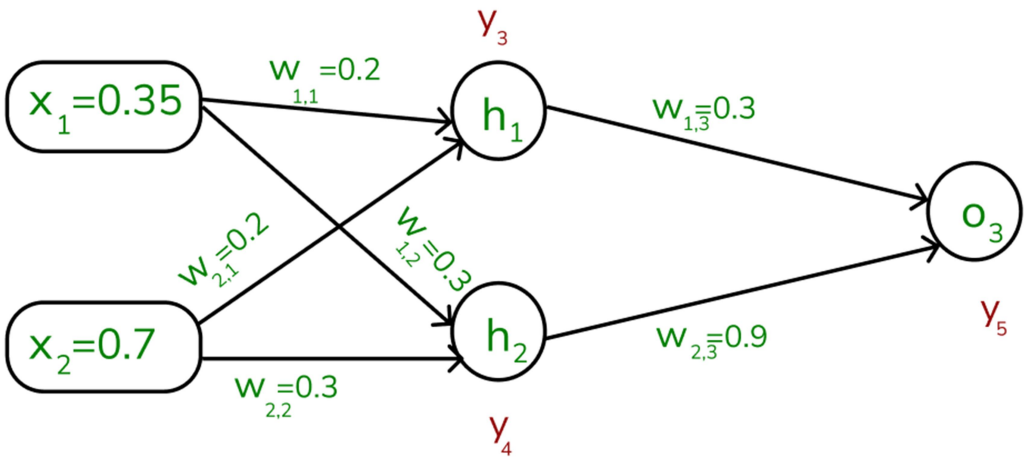
o (output): After applying the activation function to a, we get the output of the neuron:

$$ojoj = \text{activation function}(ajaj)$$

2. Sigmoid Function

The sigmoid function returns a value between 0 and 1, introducing non-linearity into the model.

$$yj=11+e-ajyj=1+e-aj1$$



To find the outputs of y3, y4 and y5

3. Computing Outputs

At h1 node

a1=(w1,1x1)+(w2,1x2)=(0.2∗0.35)+(0.2∗0.7)=0.21$a1=(w1,1x1)+(w2,1x2$ $)=(0.2∗0.35)+(0.2∗0.7)=0.21$

Once we calculated the a1 value, we can now proceed to find the y3 value:

yj=F(aj)=11+e−a1$yj=F(aj)=1+e−a11$

y3=F(0.21)=11+e−0.21$y3=F(0.21)=1+e−0.211$

y3=0.56$y3=0.56$

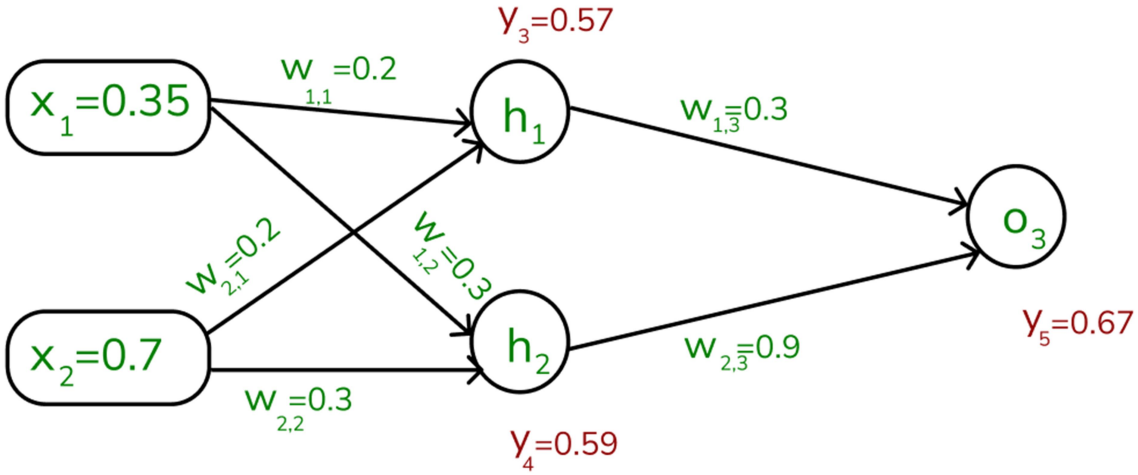
Similarly find the values of y4 at **h2** and y5 at O3

a2=(w1,2∗x1)+(w2,2∗x2)=(0.3∗0.35)+(0.3∗0.7)=0.315$a2=(w1,2∗x1)+(w2,2∗x2$ $)=(0.3∗0.35)+(0.3∗0.7)=0.315$

y4=F(0.315)=11+e−0.315$y4=F(0.315)=1+e−0.3151$

a3=(w1,3∗y3)+(w2,3∗y4)=(0.3∗0.57)+(0.9∗0.59)=0.702$a3=(w1,3∗y3)+(w2,3∗y4$ $)=(0.3∗0.57)+(0.9∗0.59)=0.702$

y5=F(0.702)=11+e−0.702=0.67$y5=F(0.702)=1+e−0.7021=0.67$



Values of y3, y4 and y5

4. Error Calculation

Our actual output is 0.5 but we obtained 0.67. To calculate the error we can use the below formula:

Errorj=ytarget−y5*Errorj*=*ytarget*−*y*5

Error=0.5−0.67=−0.17*Error*=0.5−0.67=−0.17

Using this error value we will be backpropagating.

Backpropagation

1. Calculating Gradients

The change in each weight is calculated as:

Δwij=η×δj×Oj*Δwij*=*η*×*δj*×*Oj*

Where:

- δj*δj* is the error term for each unit,
- η*η* is the learning rate.

2. Output Unit Error

For O3:

δ5=y5(1−y5)(ytarget−y5)*δ*5=*y*5(1−*y*5)(*ytarget*−*y*5)

=0.67(1−0.67)(−0.17)=−0.0376=0.67(1−0.67)(−0.17)=−0.0376

3. Hidden Unit Error

For h1:

δ3=y3(1−y3)(w1,3×δ5)*δ*3=*y*3(1−*y*3)(*w*1,3×*δ*5)

=0.56(1−0.56)(0.3×−0.0376)=−0.0027=0.56(1−0.56)(0.3×−0.0376)=−0.00

27

For h2:

δ4=y4(1−y4)(w2,3×δ5)*δ*4=*y*4(1−*y*4)(*w*2,3×*δ*5)

=0.59(1−0.59)(0.9×−0.0376)=−0.0819=0.59(1−0.59)(0.9×−0.0376)=−0.0819

4. Weight Updates

For the weights from hidden to output layer:

Δw2,3=1×(−0.0376)×0.59=−0.022184Δ*w*2,3

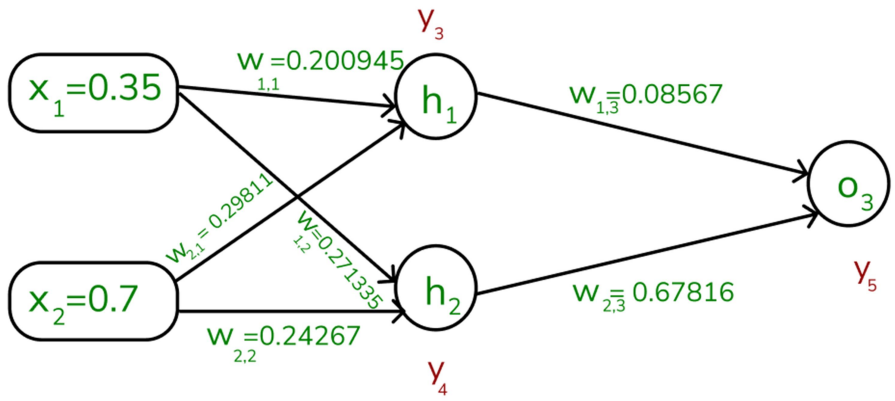
=1×(−0.0376)×0.59=−0.022184

New weight:

w2,3(new)=−0.022184+0.9=0.877816*w*2,3
(new)=−0.022184+0.9=0.877816
For weights from input to hidden layer:
Δw1,1=1×(−0.0027)×0.35=0.000945Δ*w*1,1=1×(−0.0027)×0.35=0.000945
New weight:
w1,1(new)=0.000945+0.2=0.200945*w*1,1(new)=0.000945+0.2=0.200945
Similarly other weights are updated:

- w1,2(new)=0.273225*w*1,2(new)=0.273225
- w1,3(new)=0.086615*w*1,3(new)=0.086615
- w2,1(new)=0.269445*w*2,1(new)=0.269445
- w2,2(new)=0.18534*w*2,2(new)=0.18534

The updated weights are illustrated below



Through backward pass the weights are updated

After updating the weights the forward pass is repeated yielding:

- y3=0.57*y*3=0.57
- y4=0.56*y*4=0.56
- y5=0.61*y*5=0.61

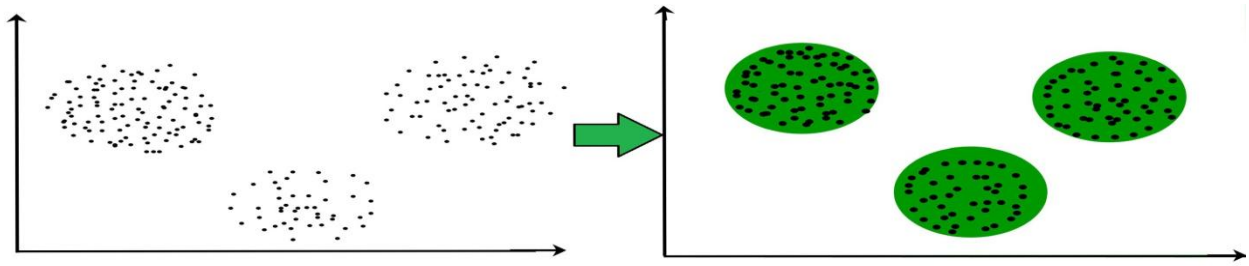
Since y5=0.61*y*5=0.61 is still not the target output the process of calculating the error and backpropagating continues until the desired output is reached.
This process demonstrates how backpropagation iteratively updates weights by minimizing errors until the network accurately predicts the output.
Error=ytarget−y5*Error*=*ytarget*−*y*5
=0.5−0.61=−0.11=0.5−0.61=−0.11
This process is said to be continued until the actual output is gained by the neural network.

UNIT-V

Clustering:

The task of **grouping data points based on their similarity with each other is called Clustering or Cluster Analysis**. This method is defined under the branch of unsupervised learning, which aims at gaining insights from unlabelled data points.

Think of it as you have a dataset of customers shopping habits. **Clustering can help you group customers with similar purchasing behaviors, which can then be used for targeted marketing, product recommendations, or customer segmentation**



Types of Clustering

Broadly speaking, there are 2 types of clustering that can be performed to group similar data points:

- **Hard Clustering:** In this type of clustering, each data point belongs to a cluster completely or not. For example, Let's say there are 4 data point and we have to cluster them into 2 clusters. So each data point will either belong to cluster 1 or cluster 2.

Data Points	Clusters
A	C1
B	C2
C	C2
D	C1

- **Soft Clustering:** In this type of clustering, instead of assigning each data point into a separate cluster, a probability or likelihood of that point being that cluster is evaluated. For example, Let's say there are 4 data point and we have to cluster them into 2 clusters. So we will be evaluating a probability of a data point belonging to both clusters. This probability is calculated for all data points.

Data Points	Probability of C1	Probability of C2
A	0.91	0.09
B	0.3	0.7
C	0.17	0.83
D	1	0

Partitioning of Data:

Using data partitioning techniques, a huge dataset can be divided into smaller, easier-to-manage portions. These techniques are applied in a variety of fields, including distributed systems, parallel computing, and database administration.

Why do we need Data Partitioning?

Data partitioning is essential for several reasons:

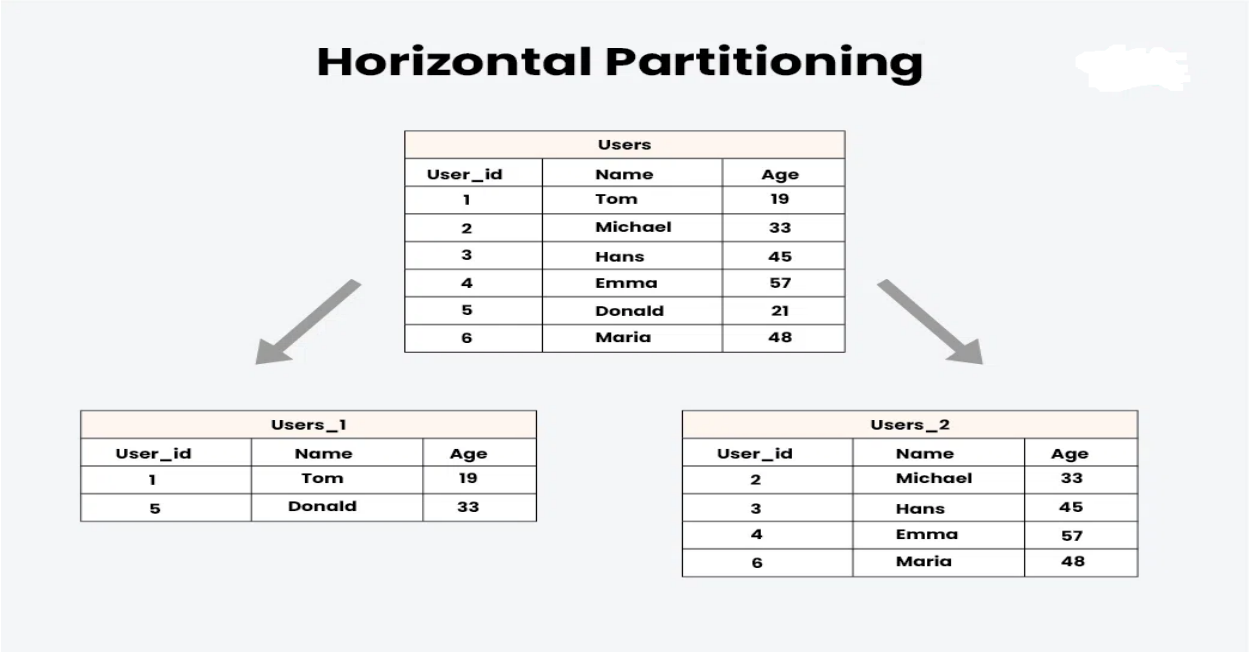
- **Performance Improvement**: By breaking data into smaller segments, systems can access only the relevant partitions, leading to faster query execution and reduced load times.
- **Scalability**: As datasets grow, partitioning allows for easier management and distribution across multiple servers or storage systems, enabling horizontal scaling.
- **Efficient Resource Utilization**: It helps optimize the use of resources by allowing systems to focus processing power on specific partitions rather than the entire dataset.
- **Enhanced Manageability**: Smaller partitions are easier to back up, restore, and maintain, facilitating better data governance and maintenance practices.

Methods of Data Partitioning

Below are the main methods of Data Partitioning:

1. Horizontal Partitioning/Sharding

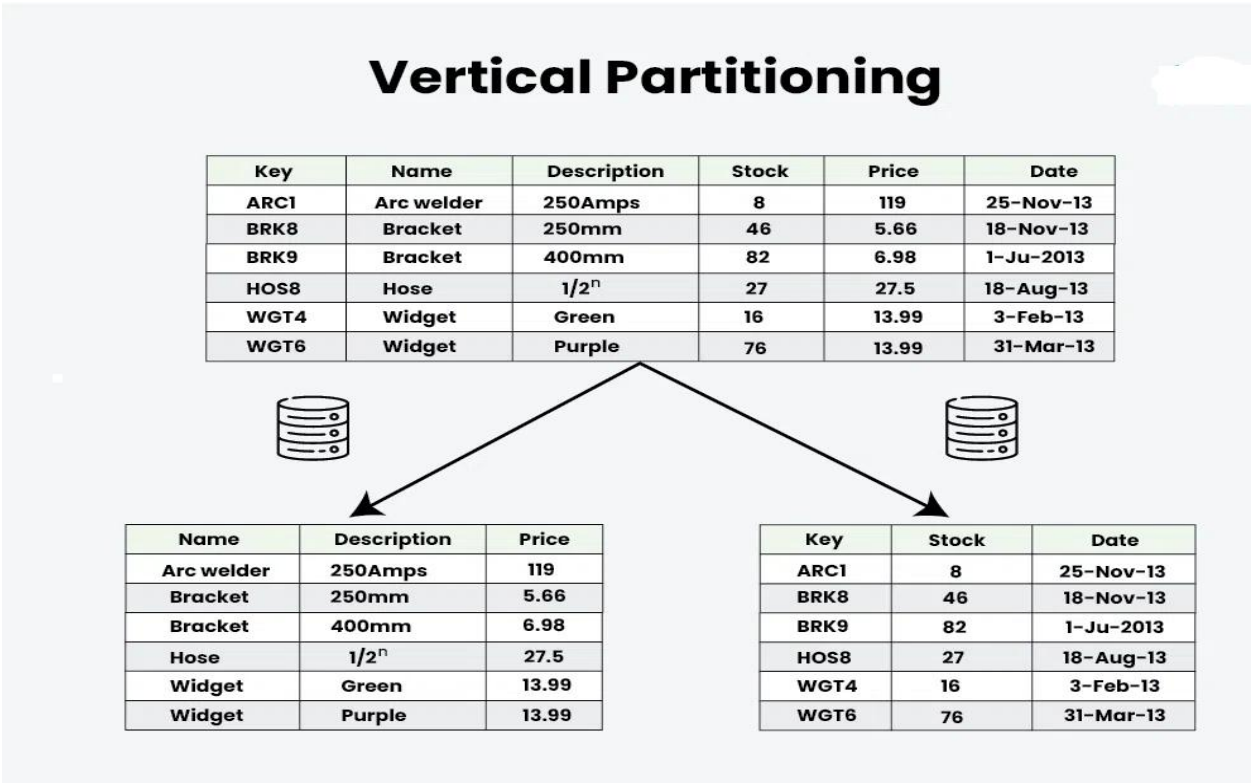
In this technique, the dataset is divided based on rows or records. Each partition contains a subset of rows, and the partitions are typically distributed across multiple servers or storage devices. Horizontal partitioning is often used in distributed databases or systems to improve parallelism and enable load balancing. Horizontal Partitioning



- **Advantages of Horizontal Partitioning/Sharding**
  - **Greater scalability:** Large datasets can be processed in parallel thanks to horizontal partitioning, which divides data among multiple computers or storage devices.
  - **Load balancing:** Data partitioning allows for the equitable distribution of workload across multiple nodes, preventing bottlenecks and improving system performance.
  - **Data separation:** Data isolation and fault tolerance are enhanced since each partition can be controlled separately. Even if one partition fails, the others can continue to function.
- **Disadvantages of Horizontal Partitioning/Sharding**
  - **Join operations:** Horizontal partitioning can make join operations across multiple partitions more complex and potentially slower, as data needs to be fetched from different nodes.
  - **Data skew:** If the distribution of data is uneven or if some partitions receive more queries or updates than others, it can result in data skew, impacting performance and load balancing.

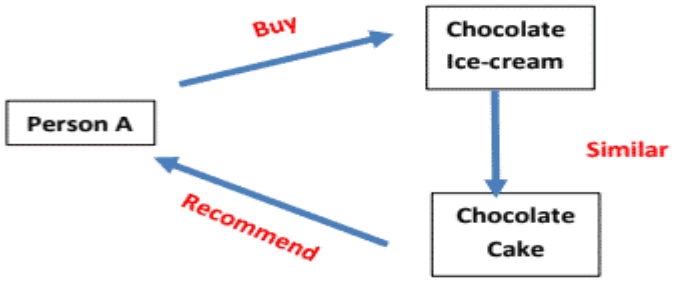
2. Vertical Partitioning

Vertical partitioning separates the dataset according to columns or attributes, in contrast to horizontal partitioning. Each partition in this method has a subset of columns for every row. When certain columns are visited more frequently than others or when different columns have different access patterns, vertical partitioning might be helpful.



Matrix Factorization | Clustering of Patterns:
This mathematical model helps the system split an entity into multiple smaller entries, through an ordered rectangular array of numbers or functions, to discover the features or information underlying the interactions between users and items. his approach recommends items based on user preferences. It matches the requirement, considering the past actions of the user, patterns detected, or any explicit feedback provided by the user, and accordingly, makes a recommendation.

Example: If you prefer the chocolate flavor and purchase a chocolate ice cream, the next time you raise a query, the system shall scan for options related to chocolate, and then, recommend you to try a chocolate cake.



How does the System make recommendations?

Let us take an example. To purchase a car, in addition to the brand name, people check for features available in the car, most common ones being safety, mileage, or aesthetic value. Few buyers consider the automatic gearbox, while others opt for a combination of two or more features. To understand this concept, let us consider a two-dimensional vector with the features of safety and mileage.

Car features	Car A	Car B	Car C	Car D
Safety	4	1	2	1
Mileage	1	4	2	2

Individual preferences	*Safety*	*Mileage*				
Person A	1	0		4	1	2	1
Person B	0	1		1	4	2	?
Person C	1	0		4	1	?	1
Person D	1	1		?	5	4	3

Divisive Clustering:

Divisive clustering **starts with one, all-inclusive cluster**. At each step, it **splits a cluster until each cluster contains a point** (or there are k clusters).

Divisive Clustering Example

The following is an example of Divisive Clustering.

Distance	a	b	c	d	e
a	0	2	6	10	9
b	2	0	5	9	8
c	6	5	0	4	5
d	10	9	4	0	3
e	9	8	5	3	0

Step 1. Split whole data into 2 clusters

1. Who hates other members the most? (in Average)
   - aa to others: mean(2,6,10,9)=6.75 →amean(2,6,10,9)=6.75 →a goes out! (Divide aa into a new cluster)
   - bb to others: mean(2,5,9,8)=6.0mean(2,5,9,8)=6.0
   - cc to others: mean(6,5,4,5)=5.0mean(6,5,4,5)=5.0
   - dd to others: mean(10,9,4,3)=6.5mean(10,9,4,3)=6.5
   - ee to others: mean(9,8,5,3)=6.25mean(9,8,5,3)=6.25
2. Everyone in the old party asks himself: *"In average, do I hate others in old party more than hating the members in the new party?"*
   - If the answer is "Yes", then he will also go to the new party.

	α=α=**distance to the old party**	β=β=**distance to the new party**	α−βα−β
b	5+9+83=7.335+9+83=7.33	2	>0>0 (bb also goes out!)
c	5+4+53=4.675+4+53=4.67	6	<0<0
d	9+4+33=5.339+4+33=5.33	10	<0<0

	α=α=**distance to the old party**	β=β=**distance to the new party**	α−βα−β
e	8+5+33=5.338+5+33=5.33	9	<0<0

3. Everyone in the old party ask himself the same question as above again and again until everyone's got the answer "No".

	α=α=**distance to the old party**	β=β=**distance to the new party**	α−βα−β
c	…	…	<0<0
d	…	…	<0<0
e	…	…	<0<0

Step 2. Choose a current cluster and split it as in **Step 1.**

1. Choose a current cluster
   - If split the cluster with the largest number of members, then the cluster {c,d,ec,d,e} will be split.
   - If split the cluster with the largest diameter, then the cluster {c,d,ec,d,e} will be split.

cluster	**diameter**
{a,b}	2
{c,d,e}	5

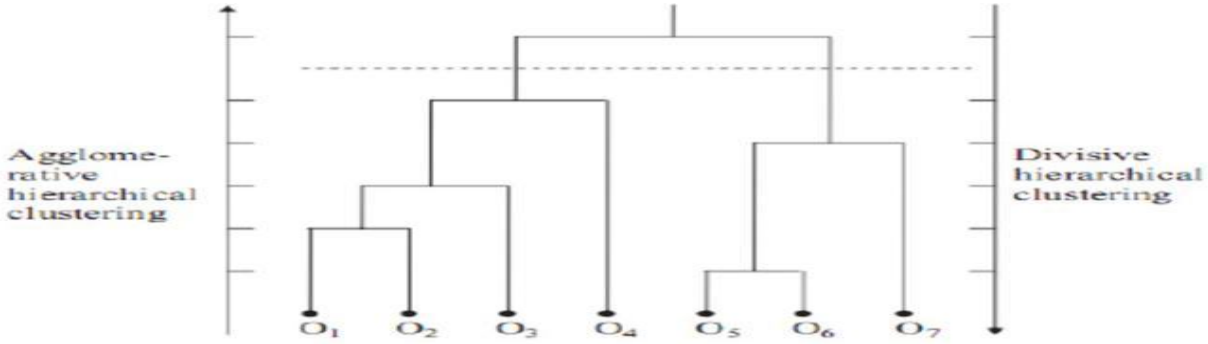
2. Split the chosen cluster as in **Step 1.**

Step 3. Repeat **Step 2.** until each cluster contains a point (or there are kk clusters)

Agglomerative Clustering:

Clustering is the broad set of techniques for finding subgroups or clusters on the basis of characterization of objects within dataset such that objects with groups are similar but different from the object of other groups. **Primary guideline of clustering is** that data inside a cluster should be very similar to each other but very different from those outside clusters. There are different types of clustering techniques like **Partitioning Methods, Hierarchical Methods and Density Based Methods.**

Method	Characteristics
Partitioning Method	• Uses mean/mediod to represent cluster centre • Adopts distance-based approach to refine clusters • Finds mutually exclusive clusters of spherical / nearly spherical shape • Effective for datasets of small - medium age
Hierarchical Method	• Creates tree-like structure through decomposition • Uses distance between nearest / furthest points in neighbouring clusters for refinement • Error can't be corrected at subsequent levels
Density Based Method	• Useful for identifying arbitrarily shaped clusters • May filter out outliers

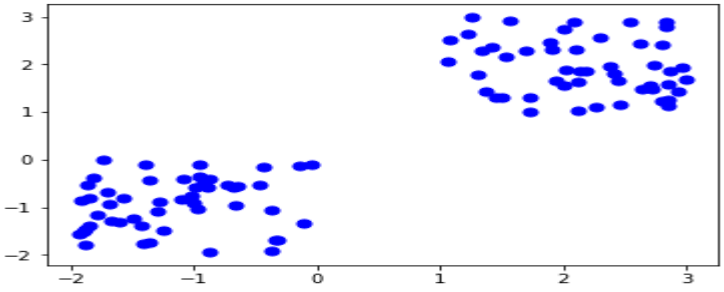


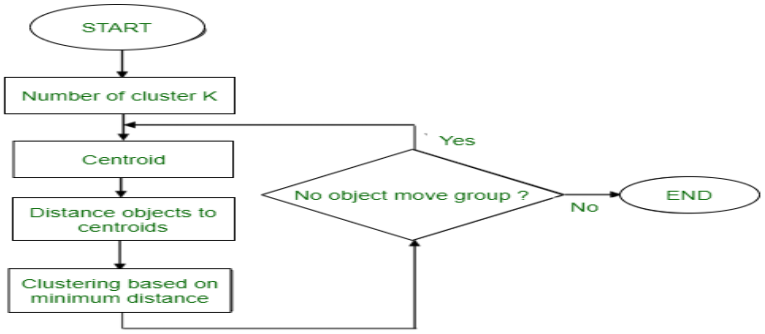
Partitional Clustering,K-means:

Partitional clustering, also known as partitioning clustering, is a type of clustering algorithm that divides a dataset into a predefined number of clusters (k).

Method:

1. Randomly assign K objects from the dataset(D) as cluster centres(C)
2. (Re) Assign each object to which object is most similar based upon mean values.
3. Update Cluster means, i.e., Recalculate the mean of each cluster with the updated values.
4. Repeat Step 2 until no change occurs.





Example: Suppose we want to group the visitors to a website using just their age as follows:

16, 16, 17, 20, 20, 21, 21, 22, 23, 29, 36, 41, 42, 43, 44, 45, 61, 62, 66

Initial Cluster:

K=2
Centroid(C1) = 16 [16]
Centroid(C2) = 22 [22]

Note: These two points are chosen randomly from the dataset.

Iteration-1:

C1 = 16.33 [16, 16, 17]
C2 = 37.25 [20, 20, 21, 21, 22, 23, 29, 36, 41, 42, 43, 44, 45, 61, 62, 66]

Iteration-2:

C1 = 19.55 [16, 16, 17, 20, 20, 21, 21, 22, 23]
C2 = 46.90 [29, 36, 41, 42, 43, 44, 45, 61, 62, 66]

Iteration-3:

C1 = 20.50 [16, 16, 17, 20, 20, 21, 21, 22, 23, 29]
C2 = 48.89 [36, 41, 42, 43, 44, 45, 61, 62, 66]

Iteration-4:

C1 = 20.50 [16, 16, 17, 20, 20, 21, 21, 22, 23, 29]
C2 = 48.89 [36, 41, 42, 43, 44, 45, 61, 62, 66]

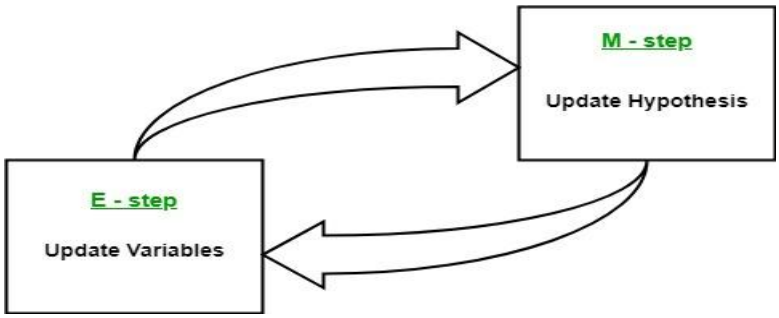
No change Between Iteration 3 and 4, so we stop. Therefore we get the clusters **(16-29)** and **(36-66)** as 2 clusters we get using K Mean Algorithm.

Expectation Maximization-Based Clustering:

Expectation-Maximization (EM) algorithm is a **iterative method** used in unsupervised machine learning to find unknown values in statistical models. **It helps to find the best values for unknown parameters especially when some data is missing or hidden.** It works in two steps:

- **E-step (Expectation Step):** Estimates missing or hidden values using current parameter estimates.
- **M-step (Maximization Step):** Updates model parameters to maximize the likelihood based on the estimated values from the E-step.

This process repeats until the model reaches a stable solution as it improve accuracy with each iteration. It is widely used in clustering like **Gaussian Mixture Models** and handling missing data.



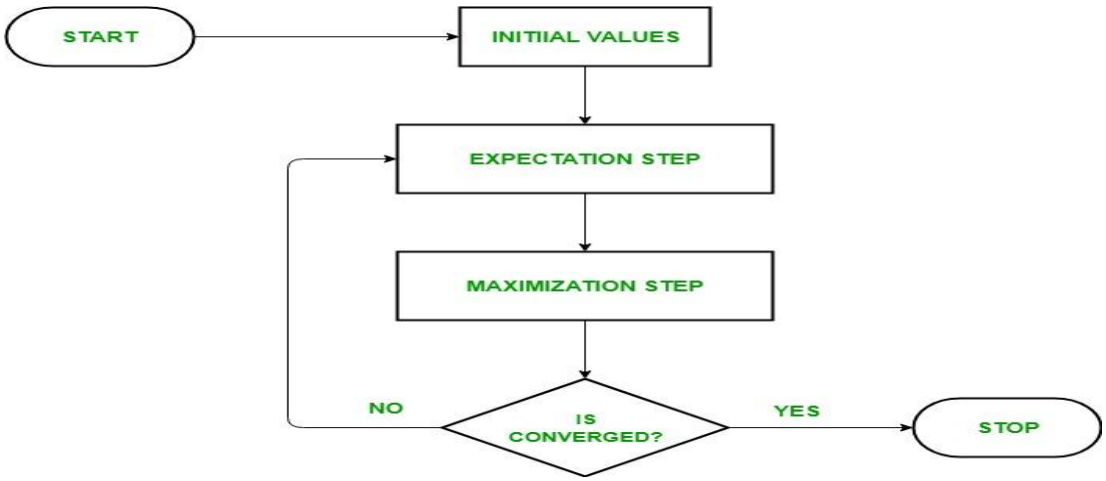
Key Terms in Expectation-Maximization (EM) Algorithm

Lets understand about some of the most commonly used key terms in the Expectation-Maximization (EM) Algorithm:

- **Latent Variables:** These are hidden parts of the data that we can't see directly but they still affect what we do see. We try to guess their values using the visible data.
- **Likelihood:** This refers to the probability of seeing the data we have based on certain assumptions or parameters. The EM algorithm tries to find the best parameters that make the data most likely.
- **Log-Likelihood:** This is just the natural log of the likelihood function. It's used to make calculations easier and measure how well the model fits the data. The EM algorithm tries to maximize the log-likelihood to improve the model fit.
- **Maximum Likelihood Estimation (MLE):** This is a method to find the best values for a model's settings called parameters. It looks for the values that make the data we observed most likely to happen.
- **Posterior Probability:** In Bayesian methods this is the probability of the parameters given both prior knowledge and the observed data. In EM it helps estimate the "best" parameters when there's uncertainty about the data.
- **Expectation (E) Step:** In this step the algorithm estimates the missing or hidden information (latent variables) based on the observed data and current parameters. It calculates probabilities for the hidden values given what we can see.
- **Maximization (M) Step:** This step update the parameters by finding the values that maximize the likelihood based on the estimates from the E-step.
- **Convergence:** Convergence happens when the algorithm has reached a stable point. This is checked by seeing if the changes in the model's parameters or the log-likelihood are small enough to stop the process.

Working of Expectation-Maximization (EM) Algorithm

So far, we've discussed the key terms in the EM algorithm. Now, let's dive into how the EM algorithm works. Here's a step-by-step breakdown of the process:



1. Initialization: The algorithm starts with initial parameter values and assumes the observed data comes from a specific model.

2. E-Step (Expectation Step):

- Find the missing or hidden data based on the current parameters.
- Calculate the posterior probability of each latent variable based on the observed data.
- Compute the log-likelihood of the observed data using the current parameter estimates.

3. M-Step (Maximization Step):

- Update the model parameters by maximize the log-likelihood.
- The better the model the higher this value.

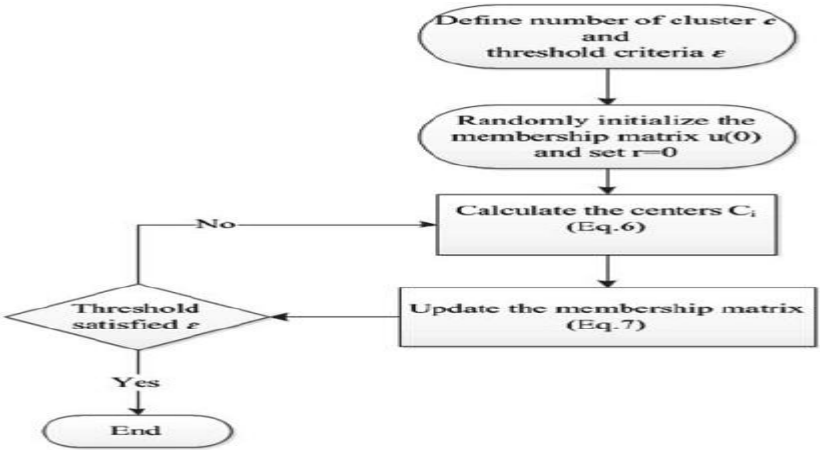
4. Convergence:

- Check if the model parameters are stable and converging.
- If the changes in log-likelihood or parameters are below a set threshold, stop. If not repeat the E-step and M-step until convergence is reached

Fuzzy C-Means Clustering:

It is an unsupervised clustering algorithm that permits us to build a fuzzy partition from data. The algorithm depends on a parameter m which corresponds to the degree of fuzziness of the solution. Large values of m will blur the classes and all elements tend to belong to all clusters. The solutions of the optimization problem depend on the parameter m. That is, different selections of m will typically lead to different partitions. Given below is a gif that shows the effect of the selection of m obtained from the fuzzy c-means.

Steps:



1. **Assume** a fixed number of clusters *k.*
2. **Initialization**: Randomly initialize the k-means μk associated with the clusters and compute the probability that each data point xi is a member of a given cluster *k*, P(point xi has label k|xi, k).

3. **Iteration**: Recalculate the centroid of the cluster as the weighted centroid given the probabilities of membership of all data points xi:

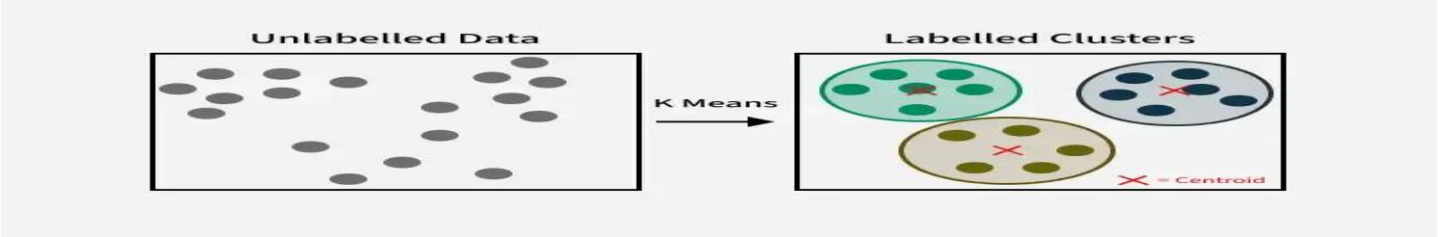
$$\mu_k(n+1) = \frac{\sum_{x_i \in k} x_i * P(\mu_k | x_i)^b}{\sum_{x_i \in k} P(\mu_k | x_i)^b}$$

4. **Termination**: Iterate until convergence or until a user-specified number of iterations has been reached (the iteration may be trapped at some local maxima or minima).

K-Means Clustering Algorithm:

K-Means Clustering is an Unsupervised Machine Learning algorithm which groups unlabeled dataset into different clusters. It is used to organize data into **groups based on their similarity**.
We are given a data set of items with certain features and values for these features like a vector. The task is to categorize those items into groups. To achieve this we will use the K-means algorithm. 'K' in the name of the algorithm represents the number of groups/clusters we want to classify our items into.

1.



. First we randomly initialize k points called means or cluster centroids.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The "points" mentioned above are called means because they are the mean values of the items categorized in them. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set. For example for a feature *x* the items have values in [0,3] we will initialize the means with values for x at [0,3].