Unit -1 Introduction to Machine Learning

Machine Learning is a subdomain of artificial intelligence. It allows computers to learn and improve from experience without being explicitly programmed by programmers, and It is designed in such a way that allows systems to identify patterns, make predictions, and make decisions based on data. Here, Python, a versatile programming language, has become a good-to-go choice for all to start with, and it helps many machine learning enthusiasts due to Pythons simplicity, a vast collection of libraries, and a large number of applications.

Artificial Intelligence, Machine Learning, and Deep Learning

Always, new technologies designed to make our lives easier are constantly being introduced. One of these critical pieces of technology is AI or artificial intelligence. You may be familiar with AI's most common form, digital assistants, which you can find on your phone and in your home.

But what is artificial intelligence? How does artificial intelligence relate to machine learning and deep learning? Are they the same? Are the terms used interchangeably or unrelated?

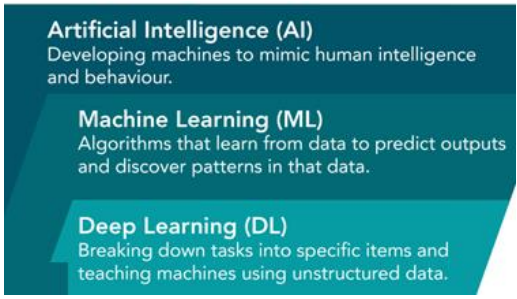
Artificial Intelligence is often used as a catch-all term for machine learning and deep learning. However, there are many differences between these types of AI, so it's essential to learn what each term represents and the differences/relationships they share.

Let's start by looking at some basic definitions of these terms:

- **Artificial Intelligence (AI)**: Developing machines to mimic human intelligence and behavior.
- **Machine Learning (ML)**: Algorithms that learn from structured data to predict outputs and discover patterns in that data.
- **Deep Learning (DL)**: Algorithms based on highly complex neural networks that mimic the way a human brain works to detect patterns in large unstructured data sets.

The Relationship Between AI, ML and DL

To begin, we're going to start with the relationship between Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL). Since these three elements are interconnected (meaning you can't have one without the other), exploring their relationship is vital before we break down each factor individually.



Machine Learning is a sub-category of AI, and Deep Learning is a sub-category of ML, meaning they are both forms of AI.

Artificial intelligence is the broad idea that machines can intelligently execute tasks by mimicking human behaviors and thought processes.

Machine learning, a subset of AI, revolves around the idea that machines can learn and adapt through experiences and data to complete specific tasks. An example would be predicting the weather forecast for the next seven days based on data from

the previous year and the previous week. Every day, the data from the previous year/week changes, so the ML model must adapt to the new data.

Deep learning is a subset of ML. DL models are based on highly complex neural networks that mimic how the brain works. With many layers of processing units, deep learning takes it a step further to learn complex patterns in large amounts of data. For example, deep learning (combined with computer vision) in a driverless car can identify a person crossing the road.

Artificial Intelligence	Machine Learning	Deep Learning
AI stands for Artificial Intelligence, and is basically the study/process which enables machines to mimic human behaviour through particular algorithm.	ML stands for Machine Learning, and is the study that uses statistical methods enabling machines to improve with experience.	DL stands for Deep Learning, and is the study that makes use of Neural Networks(similar to neurons present in human brain) to imitate functionality just like a human brain.
AI is the broader family consisting of ML and DL as it's components.	ML is the subset of AI.	DL is the subset of ML.
AI is a computer algorithm which exhibits intelligence through decision making.	ML is an AI algorithm which allows system to learn from data.	DL is a ML algorithm that uses deep(more than one layer) neural networks to analyse data and provide output accordingly.
Search Trees and much complex math is involved in AI.	If you have a clear idea about the logic(math) involved in behind and you can visualize the complex functionalities like K-Mean, Support Vector Machines, etc., then it defines the ML aspect.	If you are clear about the math involved in it but don't have idea about the features, so you break the complex functionalities into linear/lower dimension features by adding more layers, then it defines the DL aspect.
The aim is to basically increase chances of success and not accuracy.	The aim is to increase accuracy not caring much about the success ratio.	It attains the highest rank in terms of accuracy when it is trained with large amount of data.
Three broad categories/types Of AI are: Artificial Narrow Intelligence (ANI), Artificial General Intelligence (AGI) and Artificial Super Intelligence (ASI)	Three broad categories/types Of ML are: Supervised Learning, Unsupervised Learning and Reinforcement Learning	DL can be considered as neural networks with a large number of parameters layers lying in one of the four fundamental network architectures: Unsupervised Pre-trained Networks, Convolutional Neural Networks, Recurrent Neural Networks and Recursive Neural Networks

The efficiency Of AI is basically the efficiency provided by ML and DL respectively.	Less efficient than DL as it can't work for longer dimensions or higher amount of data.	More powerful than ML as it can easily work for larger sets of data.
Examples of AI applications include: Google's AI-Powered Predictions, Ridesharing Apps Like Uber and Lyft, Commercial Flights Use an AI Autopilot, etc.	Examples of ML applications include: Virtual Personal Assistants: Siri, Alexa, Google, etc., Email Spam and Malware Filtering.	Examples of DL applications include: Sentiment based news aggregation, Image analysis and caption generation, etc.
AI refers to the broad field of computer science that focuses on creating intelligent machines that can perform tasks that would normally require human intelligence, such as reasoning, perception, and decision-making.	ML is a subset of AI that focuses on developing algorithms that can learn from data and improve their performance over time without being explicitly programmed.	DL is a subset of ML that focuses on developing deep neural networks that can automatically learn and extract features from data.
AI can be further broken down into various subfields such as robotics, natural language processing, computer vision, expert systems, and more.	ML algorithms can be categorized as supervised, unsupervised, or reinforcement learning. In supervised learning, the algorithm is trained on labeled data, where the desired output is known. In unsupervised learning, the algorithm is trained on unlabeled data, where the desired output is unknown.	DL algorithms are inspired by the structure and function of the human brain, and they are particularly well-suited to tasks such as image and speech recognition.
AI systems can be rule-based, knowledge-based, or data-driven.	In reinforcement learning, the algorithm learns by trial and error, receiving feedback in the form of rewards or punishments.	DL networks consist of multiple layers of interconnected neurons that process data in a hierarchical manner, allowing them to learn increasingly complex representations of the data.

Introduction to Machine Learning

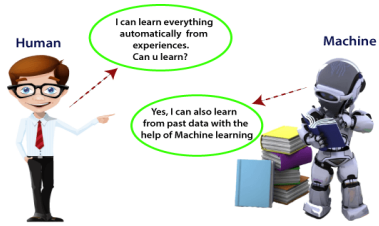
A subset of artificial intelligence known as machine learning focuses primarily on the creation of algorithms that enable a computer to independently learn from data and previous experiences. Arthur Samuel first used the term "machine learning" in 1959. It could be summarized as follows:

Without being explicitly programmed, machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things.

Machine learning algorithms create a mathematical model that, without being explicitly programmed, aids in making predictions or decisions with the assistance of sample historical data, or training data. For the purpose of developing predictive models, machine learning brings together statistics and computer science. Algorithms that learn from historical data are either constructed or utilized in machine learning. The performance will rise in proportion to the quantity of information we provide.
A machine can learn if it can gain more data to improve its performance.

What is Machine Learning?

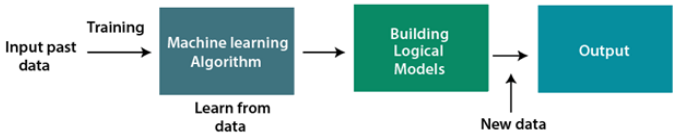
In the real world, we are surrounded by humans who can learn everything from their experiences with their learning capability, and we have computers or machines which work on our instructions. But can a machine also learn from experiences or past data like a human does? So here comes the role of **Machine Learning**.



How does Machine Learning work?

A machine learning system builds prediction models, learns from previous data, and predicts the output of new data whenever it receives it. The amount of data helps to build a better model that accurately predicts the output, which in turn affects the accuracy of the predicted output.

Let's say we have a complex problem in which we need to make predictions. Instead of writing code, we just need to feed the data to generic algorithms, which build the logic based on the data and predict the output. Our perspective on the issue has changed as a result of machine learning. The Machine Learning algorithm's operation is depicted in the following block diagram:



Features of Machine Learning:

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

Need for Machine Learning

The demand for machine learning is steadily rising. Because it is able to perform tasks that are too complex for a person to directly implement, machine learning is required. Humans are constrained by our inability to manually access vast amounts of data; as a result, we require computer systems, which is where machine learning comes in to simplify our lives.

By providing them with a large amount of data and allowing them to automatically explore the data, build models, and predict the required output, we can train machine learning algorithms. The cost function can be used to determine the amount of data and the machine learning algorithm's performance. We can save both time and money by using machine learning.

The significance of AI can be handily perceived by its utilization's cases, Presently, AI is utilized in self-driving vehicles, digital misrepresentation identification,

face acknowledgment, and companion idea by Facebook, and so on. Different top organizations, for example, Netflix and Amazon have constructed AI models that are utilizing an immense measure of information to examine the client interest and suggest item likewise.

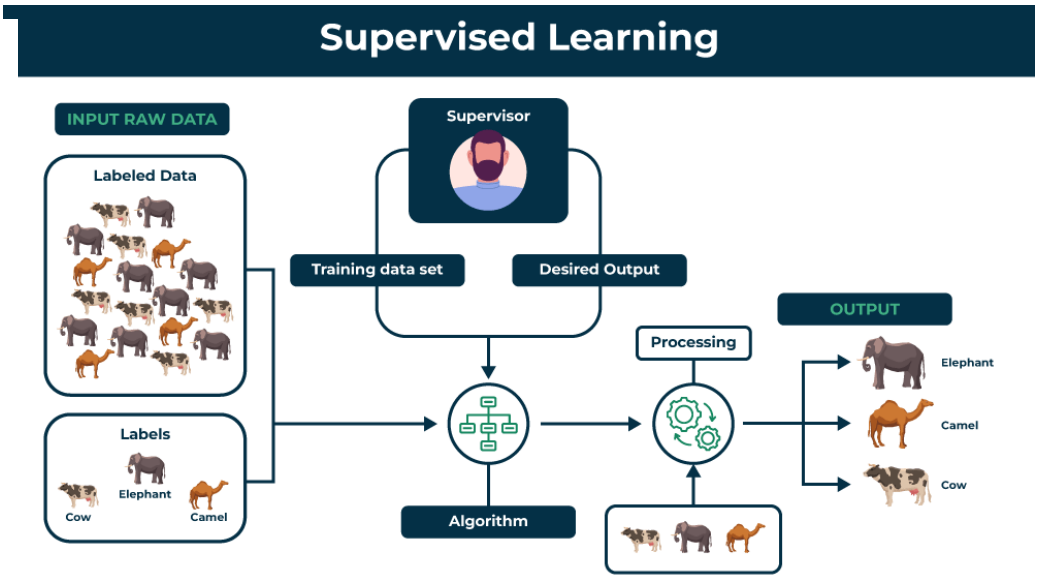
Following are some key points which show the importance of Machine Learning:

- Rapid increment in the production of data
- Solving complex problems, which are difficult for a human
- Decision making in various sector including finance
- Finding hidden patterns and extracting useful information from data.

Types of Machine Learning

At a broad level, machine learning can be classified into three types:

1. **Supervised Learning**



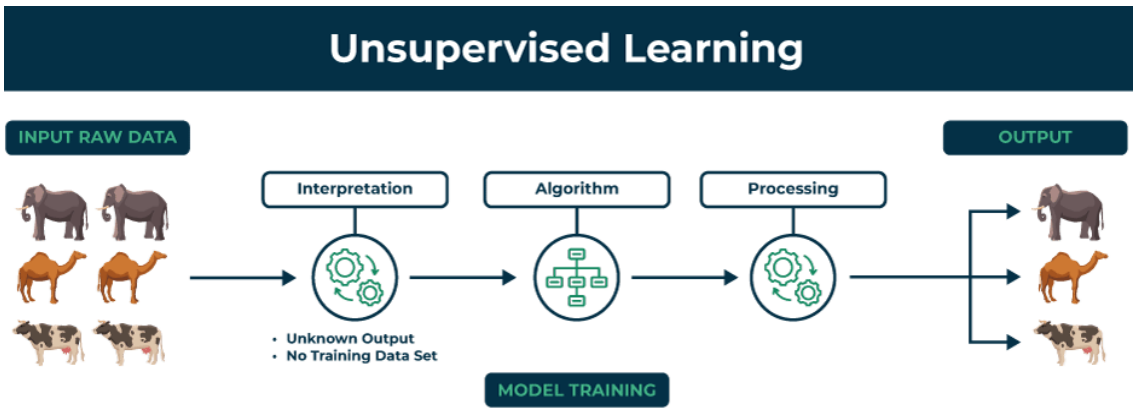
In supervised learning, sample labeled data are provided to the machine learning system for training, and the system then predicts the output based on the training data. The system uses labeled data to build a model that understands the datasets and learns about each one. After the training and processing are done, we test the model with sample data to see if it can accurately predict the output.

The mapping of the input data to the output data is the objective of supervised learning. The managed learning depends on oversight, and it is equivalent to when an understudy learns things in the management of the educator. Spam filtering is an example of supervised learning.

Supervised learning can be grouped further in two categories of algorithms:

- **Classification**
- **Regression**

2. **Unsupervised Learning**



Unsupervised learning is a learning method in which a machine learns without any supervision.

The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.

In unsupervised learning, we don't have a predetermined result. The machine tries to find useful insights from the huge amount of data. It can be further classifieds into two categories of algorithms:

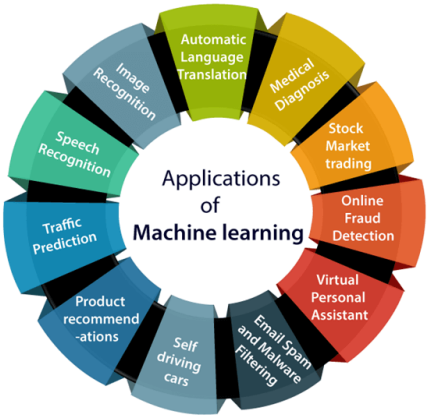
- **Clustering**
- **Association**

3. **Reinforcement Learning**

Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.

The robotic dog, which automatically learns the movement of his arms, is an example of Reinforcement learning.

Applications of Machine learning



Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:

1. Image Recognition:

Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, **Automatic friend tagging suggestion**:

Facebook provides us a feature of auto friend tagging suggestion. Whenever we upload a photo with our Facebook friends, then we automatically get a tagging suggestion with name, and the technology behind this is machine learning's **face detection** and **recognition algorithm**.

It is based on the Facebook project named "**Deep Face**," which is responsible for face recognition and person identification in the picture.

2. Speech Recognition

While using Google, we get an option of "**Search by voice**," it comes under speech recognition, and it's a popular application of machine learning.

Speech recognition is a process of converting voice instructions into text, and it is also known as "**Speech to text**", or "**Computer speech recognition**." At present, machine learning algorithms are widely used by various applications of speech recognition. **Google assistant**, **Siri**, **Cortana**, and **Alexa** are using speech recognition technology to follow the voice instructions.

3. Traffic prediction:

If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

It predicts the traffic conditions such as whether traffic is cleared, slow-moving, or heavily congested with the help of two ways:

- **Real Time location** of the vehicle form Google Map app and sensors
- **Average time has taken** on past days at the same time.

Everyone who is using Google Map is helping this app to make it better. It takes information from the user and sends back to its database to improve the performance.

4. Product recommendations:

Machine learning is widely used by various e-commerce and entertainment companies such as **Amazon**, **Netflix**, etc., for product recommendation to the user. Whenever we search for some product on Amazon, then we started getting an advertisement for the same product while internet surfing on the same browser and this is because of machine learning.

Google understands the user interest using various machine learning algorithms and suggests the product as per customer interest.

As similar, when we use Netflix, we find some recommendations for entertainment series, movies, etc., and this is also done with the help of machine learning.

5. Self-driving cars:

One of the most exciting applications of machine learning is self-driving cars. Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.

6. Email Spam and Malware Filtering:

Whenever we receive a new email, it is filtered automatically as important, normal, and spam. We always receive an important mail in our inbox with the important symbol and spam emails in our spam box, and the technology behind this is Machine learning. Below are some spam filters used by Gmail:

- Content Filter
- Header filter
- General blacklists filter
- Rules-based filters
- Permission filters

Some machine learning algorithms such as **Multi-Layer Perceptron**, **Decision tree** and **Naïve Bayes classifier** are used for email spam filtering and malware detection.

7. Virtual Personal Assistant:

We have various virtual personal assistants such as **Google assistant**, **Alexa**, **Cortana**, **Siri**. As the name suggests, they help us in finding the information using our voice instruction. These assistants can help us in various ways just by our voice instructions such as Play music, call someone, open an email, Scheduling an appointment, etc.

These virtual assistants use machine learning algorithms as an important part. These assistant record our voice instructions, send it over the server on a cloud, and decode it using ML algorithms and act accordingly.

8. Online Fraud Detection:

Machine learning is making our online transaction safe and secure by detecting fraud transaction. Whenever we perform some online transaction, there may be various ways that a fraudulent transaction can take place such as **fake accounts**, **fake ids**, and **steal money** in the middle of a transaction. So to detect this, **Feed Forward Neural network** helps us by checking whether it is a genuine transaction or a fraud transaction.

For each genuine transaction, the output is converted into some hash values, and these values become the input for the next round. For each genuine transaction, there is a specific pattern which gets change for the fraud transaction hence, it detects it and makes our online transactions more secure.

9. Stock Market trading:

Machine learning is widely used in stock market trading. In the stock market, there is always a risk of up and downs in shares, so for this machine learning's **long short term memory neural network** is used for the prediction of stock market trends.

10. Medical Diagnosis:

In medical science, machine learning is used for diseases diagnoses. With this, medical technology is growing very fast and able to build 3D models that can predict the exact position of lesions in the brain.

It helps in finding brain tumors and other brain-related diseases easily.

11. Automatic Language Translation:

Nowadays, if we visit a new place and we are not aware of the language then it is not a problem at all, as for this also machine learning helps us by converting the text into our known languages. Google's GNMT (Google Neural Machine Translation) provide this feature, which is a Neural Machine Learning that translates the text into our familiar language, and it called as automatic translation.

The technology behind the automatic translation is a sequence to sequence learning algorithm, which is used with image recognition and translates the text from one language to another language.

Unit -2 Supervised Learning

Supervised Machine Learning

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

In the real-world, supervised learning can be used for **Risk Assessment, Image classification, Fraud Detection, spam filtering**, etc.

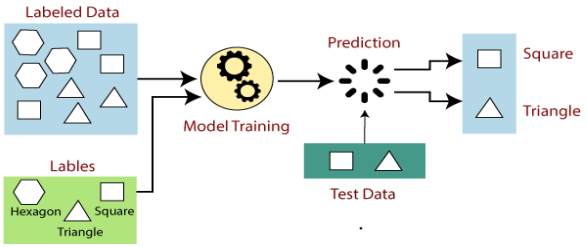
How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:

Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

- If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
- If the given shape has three sides, then it will be labelled as a **triangle**.
- If the given shape has six equal sides, then it will be labelled as **hexagon**.



Now, after training, we test our model using the test set, and the task of the model is to identify the shape.

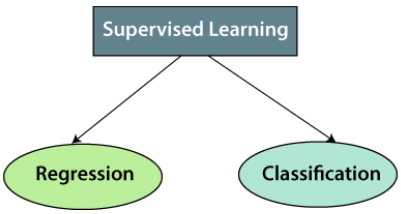
The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

Steps Involved in Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.
- Split the training dataset into training **dataset, test dataset, and validation dataset**.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.

- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

Types of supervised Machine Learning Algorithms:



Supervised learning can be further divided into two types of problems:

1. Regression

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

2. Classification

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.
Spam Filtering,

- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.
- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as **fraud detection, spam filtering**, etc.

Disadvantages of supervised learning:

- Supervised learning models are not suitable for handling the complex tasks.
- Supervised learning cannot predict the correct output if the test data is different from the training dataset.
- Training required lots of computation times.
- In supervised learning, we need enough knowledge about the classes of object.

Data Pre-processing

Pre-processing data is a crucial step in machine learning, especially for regression tasks. Raw data often contains noise, missing values, or varying scales, which can negatively impact model performance. Below are key pre-processing techniques

```
import numpy as np
from sklearn import preprocessing
#We imported a couple of packages. Let's create some sample data and add the line to this file:
input_data = np.array([[3, -1.5, 3, -6.4], [0, 3, -1.3, 4.1], [1, 2.3, -2.9, -4.3]])
```

We are now ready to operate on this data.

Pre-processing Techniques

Data can be pre-processed using several techniques as discussed here −

Mean removal

It involves removing the mean from each feature so that it is cantered on zero. Mean removal helps in removing any bias from the features. You can use the following code for mean removal

```
data_standardized = preprocessing.scale(input_data)
print "\nMean = ", data_standardized.mean(axis = 0)
print "Std deviation = ", data_standardized.std(axis = 0)
```

Now run the following command on the terminal **output**

```
Mean = [ 5.55111512e-17 -3.70074342e-17 0.00000000e+00 -1.85037171e-17]
Std deviation = [1. 1. 1. 1.]
```

Observe that in the output, mean is almost 0 and the standard deviation is 1.

Scaling

The values of every feature in a data point can vary between random values. So, it is important to scale them so that this matches specified rules.

You can use the following code for scaling −

```
data_scaler = preprocessing.MinMaxScaler(feature_range = (0, 1))
data_scaled = data_scaler.fit_transform(input_data)
print "\nMin max scaled data = ", data_scaled
```

Now run the code and you can observe the following **output**

```
Min max scaled data = [ [ 1. 0. 1. 0. ]
                        [ 0. 1. 0.27118644 1. ]
                        [ 0.33333333 0.84444444 0. 0.2 ]
]
```

That all the values have been scaled between the given range.

Normalization

Normalization involves adjusting the values in the feature vector so as to measure them on a common scale. Here, the values of a feature vector are adjusted so that they sum up to 1. We add the following lines to the prepro.py file

You can use the following code for normalization

```
data_normalized = preprocessing.normalize(input_data, norm  = 'l1')
print "\nL1 normalized data = ", data_normalized
```

Now run the code and you can observe the following **output**

```
L1 normalized data = [  [ 0.21582734 -0.10791367 0.21582734 -0.46043165]
                        [ 0. 0.35714286 -0.1547619 0.48809524]
                        [ 0.0952381 0.21904762 -0.27619048 -0.40952381]
]
```

Normalization is used to ensure that data points do not get boosted due to the nature of their features.

Binarization

Binarization is used to convert a numerical feature vector into a Boolean vector. You can use the following code for binarization

```
data_binarized = preprocessing.Binarizer(threshold=1.4).transform(input_data)
print "\nBinarized data =", data_binarized
```

Now run the code and you can observe the following **output**

```
Binarized data = [[ 1. 0. 1. 0.]
                  [ 0. 1. 0. 1.]
                  [ 0. 1. 0. 0.]
                 ]
```

This technique is helpful when we have prior knowledge of the data.

Label Encoding

In supervised learning, we mostly come across a variety of labels which can be in the form of numbers or words. If they are numbers, then they can be used directly by the algorithm. However, many times, labels need to be in readable form. Hence, the training data is usually labelled with words.

Label encoding refers to changing the word labels into numbers so that the algorithms can understand how to work on them. Let us understand in detail how to perform label encoding. Create a new Python file, and import the pre-processing package

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
input_classes = ['suzuki', 'ford', 'suzuki', 'toyota', 'ford', 'bmw']
label_encoder.fit(input_classes)
print "\nClass mapping:"
for i, item in enumerate(label_encoder.classes_):
print item, '-->', i
```

Now run the code and you can observe the following output

```
Class mapping:
bmw --> 0
ford --> 1
suzuki --> 2
toyota --> 3
```

As shown in above output, the words have been changed into 0-indexed numbers. Now, when we deal with a set of labels, we can transform them as follows –

```
labels = ['toyota', 'ford', 'suzuki']
encoded_labels = label_encoder.transform(labels)
print "\nLabels =", labels
print "Encoded labels =", list(encoded_labels)
```

Now run the code and you can observe the following output –

```
Labels = ['toyota', 'ford', 'suzuki']
Encoded labels = [3, 1, 2]
```

This is efficient than manually maintaining mapping between words and numbers. You can check by transforming numbers back to word labels as shown in the code here

```
encoded_labels = [3, 2, 0, 2, 1]
decoded_labels = label_encoder.inverse_transform(encoded_labels)
print "\nEncoded labels =", encoded_labels
print "Decoded labels =", list(decoded_labels)
```

Now run the code and you can observe the following output –

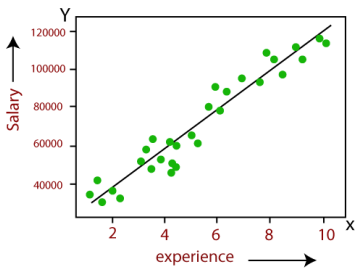
```
Encoded labels = [3, 2, 0, 2, 1]
Decoded labels = ['toyota', 'suzuki', 'bmw', 'suzuki', 'ford']
```

From the output, you can observe that the mapping is preserved perfectly.

Linear Regression:

- Linear regression is a statistical regression method which is used for predictive analysis.
- It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables.
- It is used for solving the regression problem in machine learning.
- Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.
- If there is only one input variable (x), then such linear regression is called **simple linear regression**. And if there is more than one input variable, then such linear regression is called **multiple linear regression**.
- The relationship between variables in the linear regression model can be explained using the below image. Here we are predicting the salary of an employee on the basis of **the year of experience**.
- Below is the mathematical equation for Linear regression: Y= aX+b

Here, Y = dependent variables (target variables), X= Independent variables (predictor variables), a and b are the linear coefficients



Some popular applications of linear regression are:

- **Analysing trends and sales estimates**
- **Salary forecasting**
- **Real estate prediction**
- **Arriving at ETAs in traffic.**

Classification

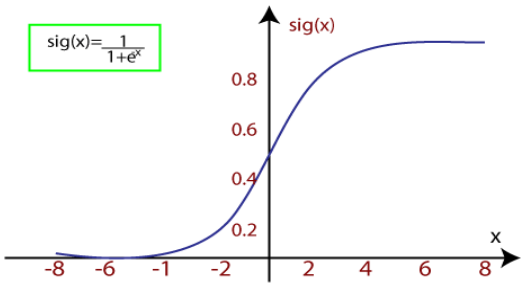
Logistic Regression:

- Logistic regression is another supervised learning algorithm which is used to solve the classification problems. In **classification problems**, we have dependent variables in a binary or discrete format such as 0 or 1.
- Logistic regression algorithm works with the categorical variable such as 0 or 1, Yes or No, True or False, Spam or not spam, etc.
- It is a predictive analysis algorithm which works on the concept of probability.
- Logistic regression is a type of regression, but it is different from the linear regression algorithm in the term how they are used.
- Logistic regression uses **sigmoid function** or logistic function which is a complex cost function. This sigmoid function is used to model the data in logistic regression. The function can be represented as:
  - f(x)= Output between the 0 and 1 value.
  - x= input to the function
  - e= base of natural logarithm.

$$f(x)= \frac{1}{1+e^{-x}}$$

When we provide the input values (data) to the function, it gives the S-curve as follows:

It uses the concept of threshold levels, values above the threshold level are rounded up to 1, and values below the threshold level are rounded up to 0.



There are three types of logistic regression:

- **Binary(0/1, pass/fail)**
- **Multi(cats, dogs, lions)**
- **Ordinal(low, medium, high)**

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, which can be described as:

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

- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

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

Where,

- **P(A|B) is Posterior probability**: Probability of hypothesis A on the observed event B.
- **P(B|A) is Likelihood probability**: Probability of the evidence given that the probability of a hypothesis is true.
- **P(A) is Prior Probability**: Probability of hypothesis before observing the evidence.
- **P(B) is Marginal Probability**: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below **example:**

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?
Solution: To solve this, first consider the below dataset:

	Outlook	Play		Outlook	Play		Outlook	Play
0	Rainy	Yes	**5**	Rainy	Yes	**10**	Sunny	Yes
1	Sunny	Yes	**6**	Sunny	Yes	**11**	Rainy	No
2	Overcast	Yes	**7**	Overcast	Yes	**12**	Overcast	Yes
3	Overcast	Yes	**8**	Rainy	No	**13**	Overcast	Yes
4	Sunny	No	**9**	Sunny	No			

Frequency table for the Weather Conditions:		
Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition			
Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)
P(Sunny|Yes)= 3/10= 0.3
P(Sunny)= 0.35
P(Yes)=0.71
So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)
P(Sunny|NO)= 2/4=0.5
P(No)= 0.29
P(Sunny)= 0.35
So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**
So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**
Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Types of Naïve Bayes Model:

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

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.

The classifier uses the frequency of words for the predictors.

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

Python Implementation of the Naïve Bayes algorithm:

Now we will implement a Naive Bayes Algorithm using Python. So for this, we will use the "**user_data**" **dataset**, which we have used in our other classification model. Therefore, we can easily compare the Naive Bayes model with the other models.

Steps to implement:

- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)
- Visualizing the test set result.

Data Pre-processing step

In this step, we will pre-process/prepare the data so that we can use it efficiently in our code. It is similar as we did in data-pre-processing. The code for this is given below:

#Importing the libraries

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('user_data.csv')
x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size
= 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

dataset - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0
16	15733883	Male	47	25000	1
17	15617482	Male	45	26000	1
18	15704583	Male	46	28000	1
19	15621083	Female	48	29000	1

In the above code, we have loaded the dataset into our program using "**dataset = pd.read_csv('user_data.csv')**. The loaded dataset is divided into training and test set, and then we have scaled the feature variable.

The output for the dataset is given as:

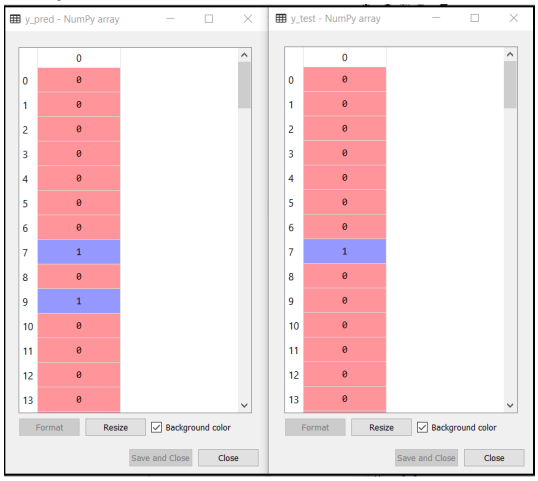
Fitting Naive Bayes to the Training Set

After the pre-processing step, now we will fit the Naive Bayes model to the Training set. Below is the code for it:

```
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
```

In the above code, we have used the **GaussianNB classifier** to fit it to the training dataset. We can also use other classifiers as per our requirement.

Output: Out[6]: GaussianNB(priors=None, var_smoothing=1e-09)



Prediction of the test set result

Now we will predict the test set result. For this, we will create a new predictor variable **y_pred**, and will use the predict function to make the predictions.

```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
```

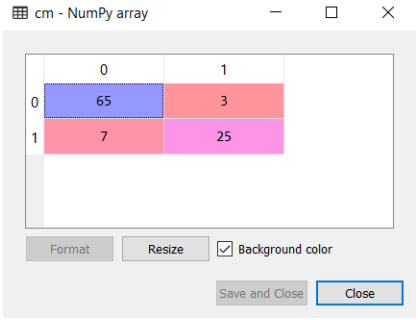
Output:

The above output shows the result for prediction vector **y_pred** and real vector y_test. We can see that some predications are different from the real values, which are the incorrect predictions.

Creating Confusion Matrix:

Now we will check the accuracy of the Naive Bayes classifier using the Confusion matrix. Below is the code for it:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```



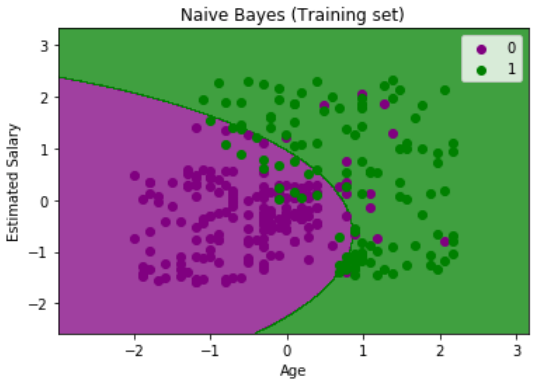
Output:

As we can see in the above confusion matrix output, there are 7+3= 10 incorrect predictions, and 65+25=90 correct predictions.

Visualizing the training set result:

Next we will visualize the training set result using Naïve Bayes Classifier. Below is the code for it:

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
 1, stop = x_set[:, 0].max() + 1, step = 0.01),
                     nm.arange(start = x_set[:, 1].min() -
 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(
), X2.ravel()]).T).reshape(X1.shape),
         alpha = 0.75, cmap = ListedColormap(('purple', 'gre
en')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```



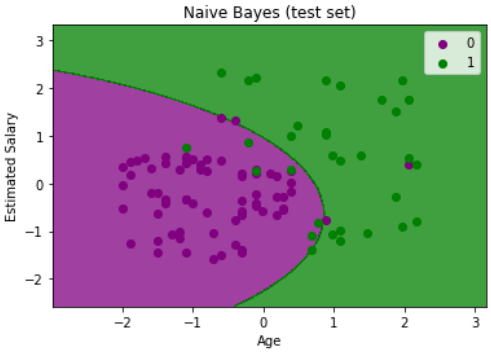
```
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

In the above output we can see that the Naïve Bayes classifier has segregated the data points with the fine boundary. It is Gaussian curve as we have used **GaussianNB** classifier in our code.

Visualizing the Test set result:

```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
 1, stop = x_set[:, 0].max() + 1, step = 0.01),
              nm.arange(start = x_set[:, 1].min() -
 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
        alpha = 0.75, cmap = ListedColormap(('purple', 'green'
)))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
         c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```



Output:

The above output is final output for test set data. As we can see the classifier has created a Gaussian curve to divide the "purchased" and "not purchased" variables. There are some wrong predictions which we have calculated in Confusion matrix. But still it is pretty good classifier.

What is Training data?

Testing data is used to determine the performance of the trained model, whereas training data is used to train the machine learning model. Training data is the power that supplies the model in machine learning, it is larger than testing data. Because more data helps to more effective predictive models. When a machine learning algorithm receives data from our records, it recognizes patterns and creates a decision-making model.

Algorithms allow a company's past experience to be used to make decisions. It analyses all previous cases and their results and, using this data creates models to score

and predict the outcome of current cases. The more data ML models have access to, the more reliable their predictions get over time.

What is Testing Data?

You will need unknown information to test your machine learning model after it was created (using your training data). This data is known as testing data, and it may be used to assess the progress and efficiency of your algorithms' training as well as to modify or optimize them for better results.

- Showing the original set of data.
- Be large enough to produce reliable projections

This dataset needs to be "unseen" and recent. This is because the training data was already "learned" by your model. You can decide if it is operating successfully or when it need more training data to fulfil your standards by observing how it performs on fresh test data. Test data provides as a last, real check if an unknown dataset was correctly trained by the machine learning algorithm.

Need of Splitting dataset into Train and Test set

Splitting the dataset into train and test sets is one of the important parts of data pre-processing, as by doing so, we can improve the performance of our model and hence give better predictability.

We can understand it as if we train our model with a training set and then test it with a completely different test dataset, and then our model will not be able to understand the correlations between the features.

Therefore, if we train and test the model with two different datasets, then it will decrease the performance of the model. Hence it is important to split a dataset into two parts, i.e., train and test set.

In this way, we can easily evaluate the performance of our model. Such as, if it performs well with the training data, but does not perform well with the test dataset, then it is estimated that the model may be over fitted.

Why do we need Training data and Testing data?

Training data teaches a machine learning model how to behave, whereas testing data assesses how well the model has learned.

- **Training Data:** The machine learning model is taught how to generate predictions or perform a specific task using training data. Since it is usually identified, every data point's output from the model is known. In order to provide predictions, the model must first learn to recognize patterns in the data. Training data can be compared to a student's textbook when learning a new subject. The learner learns by reading the text and completing the tasks, and the book offers all the knowledge they require.
- **Testing Data:** The performance of the machine learning model is measured using testing data. Usually, it is labelled and distinct from the training set. This indicates that for every data point, the model's result is unknown. On the testing data, the model's accuracy in predicting outcomes is assessed. Testing data is comparable to the exam a student takes to determine how well-versed in a subject they are. The test asks questions that the student must respond to, and the test results are used to gauge the student's comprehension.

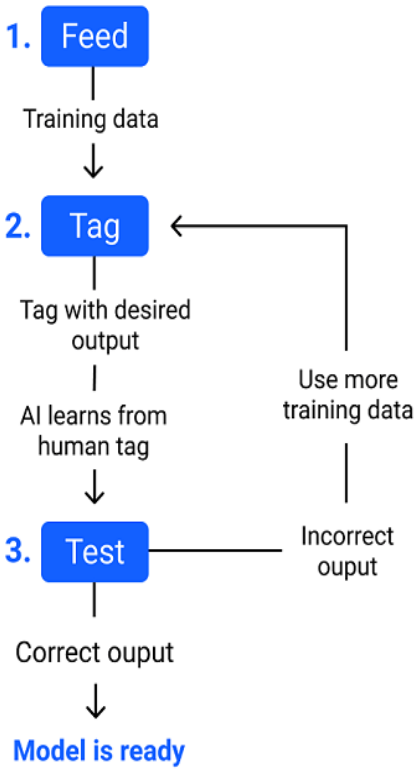
How do training and testing data work in Machine Learning?

Machine Learning algorithms enable the machines to make predictions and solve problems on the basis of past observations or experiences. These experiences or observations an algorithm can take from the training data, which is fed to it. Further, one of the great things about ML algorithms is that they can learn and improve over time on their own, as they are trained with the relevant training data.

Once the model is trained enough with the relevant training data, it is tested with the test data. We can understand the whole process of training and testing in three steps, which are as follows:

1. **Feed:** Firstly, we need to train the model by feeding it with training input data.
2. **Define:** Now, training data is tagged with the corresponding outputs (in Supervised Learning), and the model transforms the training data into text vectors or a number of data features.
3. **Test:** In the last step, we test the model by feeding it with the test data/unseen dataset. This step ensures that the model is trained efficiently and can generalize well.

The above process is explained using a flowchart given below:



Traits of Quality training data

As the ability to the prediction of an ML model highly depends on how it has been trained, therefore it is important to train the model with quality data. Further, ML works on the concept of "Garbage in, Garbage Out." It means that whatever type of data we will input into our model; it will make the predictions accordingly. For a quality training data, the below points should be considered:

1. Relevant: The very first quality of training data should be relevant to the problem that you are going to solve. It means that whatever data you are using should be relevant to the current problem. For example, if you are building a model to analyse social media data, then data should be taken from different social sites such as Twitter, Facebook, Instagram, etc.

2. Uniform: There should always be uniformity among the features of a dataset. It means all data for a particular problem should be taken from the same source with the same attributes.

3. Consistency: In the dataset, the similar attributes must always correspond to the similar label in order to ensure uniformity in the dataset.

4. Comprehensive: The training data must be large enough to represent sufficient features that you need to train the model in a better way. With a comprehensive dataset, the model will be able to learn all the edge cases.

What is cross-validation?

Cross-validation is a technique for evaluating a machine learning model and testing its performance. CV is commonly used in applied ML tasks. It helps to compare and select an appropriate model for the specific predictive modelling problem.
CV is easy to understand, easy to implement, and it tends to have a lower bias than other methods used to count the model's efficiency scores. All this makes cross-validation a powerful tool for selecting the best model for the specific task.

There are a lot of different techniques that may be used to cross-validate a model. Still, all of them have a similar algorithm:

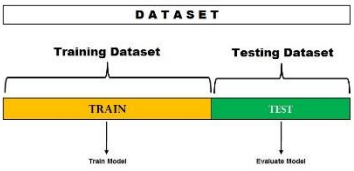
1. Divide the dataset into two parts: one for training, other for testing
2. Train the model on the training set
3. Validate the model on the test set
4. Repeat 1-3 steps a couple of times. This number depends on the CV method that you are using

there are plenty of CV techniques. Some of them are commonly used, others work only in theory. Let's see the cross-validation methods

- Hold-out
- K-folds
- Leave-one-out
- Leave-p-out
- Stratified K-folds
- Repeated K-folds
- Nested K-folds
- Time series CV

Hold-out cross-validation

Hold-out cross-validation is the simplest and most common technique. You might not know that it is a hold-out method but you certainly use it every day.
The algorithm of hold-out technique:



1. Divide the dataset into two parts: the training set and the test set. Usually, 80% of the dataset goes to the training set and 20% to the test set but you may choose any splitting that suits you better
2. Train the model on the training set
3. Validate on the test set
4. Save the result of the validation

That's it. We usually use the hold-out method on large datasets as it requires training the model only once. It is really easy to implement hold-out. For example, you may do it using sklearn.model_selection.train_test_split.

```
import numpy as np
from sklearn.model_selection import train_test_split
X, y = np.arange(10).reshape((5, 2)), range(5)
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,
random_state=111)
```

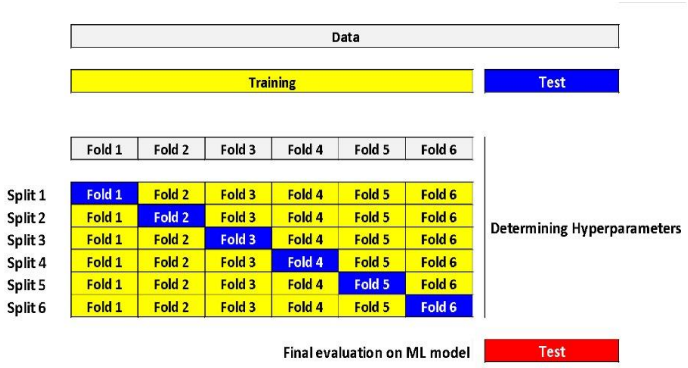
Still, hold-out has a major disadvantage. For example, a dataset that is not completely even distribution-wise. If so we may end up in a rough spot after the split. For example, the

training set will not represent the test set. Both training and test sets may differ a lot, one of them might be easier or harder.

Moreover, the fact that we test our model only once might be a bottleneck for this method. Due to the reasons mentioned before, the result obtained by the hold-out technique may be considered inaccurate.

K-Fold cross-validation

k-Fold cross-validation is a technique that minimizes the disadvantages of the hold-out method. k-Fold introduces a new way of splitting the dataset which helps to overcome the "test only once bottleneck". The algorithm of the k-Fold technique:



1. Pick a number of folds – k. Usually, k is 5 or 10 but you can choose any number which is less than the dataset's length.
2. Split the dataset into k equal (if possible) parts (they are called folds)
3. Choose k – 1 folds as the training set. The remaining fold will be the test set
4. Train the model on the training set. On each iteration of cross-validation, you must train a new model independently of the model trained on the previous iteration
5. Validate on the test set
6. Save the result of the validation
7. Repeat steps 3 – 6 k times. Each time use the remaining fold as the test set. In the end, you should have validated the model on every fold that you have.
8. To get the final score average the results that you got on step 6.

To perform k-Fold cross-validation you can use sklearn.model_selection.KFold.

```
import numpy as np
from sklearn.model_selection import KFold
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4])
kf = KFold(n_splits=2)

for train_index, test_index in kf.split(X):
  print("TRAIN:", train_index, "TEST:", test_index)
  X_train, X_test = X[train_index], X[test_index]
  y_train, y_test = y[train_index], y[test_index]
```

In general, it is always better to use k-Fold technique instead of hold-out. In a head to head, comparison k-Fold gives a more stable and trustworthy result since training and testing is performed on several different parts of the dataset. We can make the overall score even more robust if we increase the number of folds to test the model on many different sub-datasets.

Still, k-Fold method has a disadvantage. Increasing k results in training more models and the training process might be really expensive and time-consuming.

Leave-one-out cross-validation

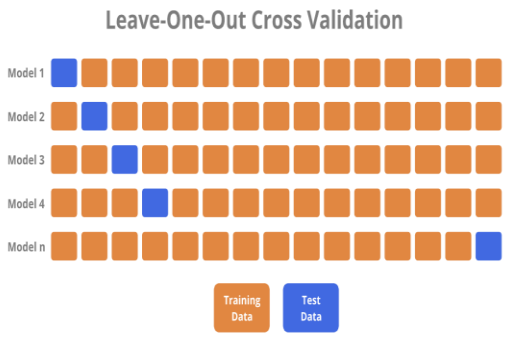
we perform training on the whole dataset but leaves only one data-point of the available dataset and then iterates for each data-point. In LOOCV, the model is trained on ***n*−1** samples and tested on the one omitted sample, repeating this process for each data point in the dataset. It has some advantages as well as disadvantages also.

An advantage of using this method is that we make use of all data points and hence it is low bias.

The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it takes a lot of execution time as it iterates over 'the number of data points' times.

The algorithm of LOOCV technique:

1. Choose one sample from the dataset which will be the test set
2. The remaining n – 1 samples will be the training set
3. Train the model on the training set. On each iteration, a new model must be trained
4. Validate on the test set
5. Save the result of the validation
6. Repeat steps 1 – 5 n times as for n samples we have n different training and test sets
7. To get the final score average the results that you got on step 5.



For LOOCV sklearn also has a built-in method. It can be found in the model_selection library – sklearn.model_selection.LeaveOneOut.

```
import numpy as np
from sklearn.model_selection import LeaveOneOut
X = np.array([[1, 2], [3, 4]])
y = np.array([1, 2])
loo = LeaveOneOut()
for train_index, test_index in loo.split(X):
   print("TRAIN:", train_index, "TEST:", test_index)
   X_train, X_test = X[train_index], X[test_index]
   y_train, y_test = y[train_index], y[test_index]
```

The greatest advantage of Leave-one-out cross-validation is that it doesn't waste much data. We use only one sample from the whole dataset as a test set, whereas the rest is the training set. But when compared with k-Fold CV, LOOCV requires building n models instead of k models, when we know that **n** which stands for the number of samples in the dataset is much higher than k. It means LOOCV is more computationally expensive than k-Fold, it may take plenty of time to cross-validate the model using LOOCV.

Thus, the Data Science community has a general rule based on empirical evidence and different researches, which suggests that 5- or 10-fold cross-validation should be preferred over LOOCV.

Leave-P-out cross-validation

Leave-p-out cross-validation (LpOC) is similar to Leave-one-out CV as it creates all the possible training and test sets by using p samples as the test set. All mentioned about LOOCV is true and for LpOC.

Still, it is worth mentioning that unlike LOOCV and k-Fold test sets will overlap for LpOC if p is higher than 1.

The p datasets are left out of the training data. It means, if there are total n data points in the original input dataset, then n-p data points will be used as the training dataset and the p data points as the validation set. This complete process is repeated for all the samples, and the average error is calculated to know the effectiveness of the model.

There is a disadvantage of this technique; that is, it can be computationally difficult for the large p. The algorithm of LpOC technique:

1. Choose p samples from the dataset which will be the test set
2. The remaining n – p samples will be the training set
3. Train the model on the training set. On each iteration, a new model must be trained
4. Validate on the test set
5. Save the result of the validation
6. Repeat steps 2 – 5 Cpn times
7. To get the final score average the results that you got on step 5
8. You can perform Leave-p-out CV using sklearn – sklearn.model_selection.LeavePOut.

```
import numpy as np
from sklearn.model_selection import LeavePOut
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 3, 4])
lpo = LeavePOut(2)

for train_index, test_index in lpo.split(X):
   print("TRAIN:", train_index, "TEST:", test_index)
   X_train, X_test = X[train_index], X[test_index]
   y_train, y_test = y[train_index], y[test_index]
```

Stratified k-Fold cross-validation

Sometimes we may face a large imbalance of the target value in the dataset. For example, in a dataset concerning wristwatch prices, there might be a larger number of wristwatch having a high price. In the case of classification, in cats and dogs' dataset there might be a large shift towards the dog class.

Stratified k-Fold is a variation of the standard k-Fold CV technique which is designed to be effective in such cases of target imbalance.

It works as follows. Stratified k-Fold splits the dataset on k folds such that each fold contains approximately the same percentage of samples of each target class as the complete set. In the case of regression, Stratified k-Fold makes sure that the mean target value is approximately equal in all the folds.

The algorithm of Stratified k-Fold technique:

1. Pick a number of folds – k
2. Split the dataset into k folds. Each fold must contain approximately the same percentage of samples of each target class as the complete set

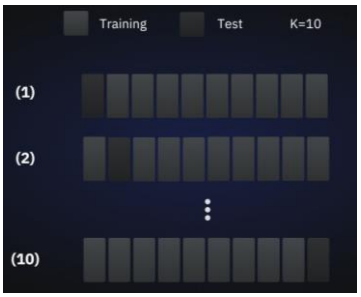
3. Choose **k – 1** folds which will be the training set. The remaining fold will be the test set
4. Train the model on the training set. On each iteration a new model must be trained
5. Validate on the test set
6. Save the result of the validation
7. Repeat steps 3 – 6 k times. Each time use the remaining fold as the test set. In the end, you should have validated the model on every fold that you have.
8. To get the final score average the results that you got on step 6.

As you may have noticed, the algorithm for Stratified k-Fold technique is similar to the standard k-Folds. You don't need to code something additionally as the method will do everything necessary for you.

Stratified k-Fold also has a built-in method in sklearn sklearn.model_selection.StratifiedKFold.

```
import numpy as np
from sklearn.model_selection import StratifiedKFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 1, 1])
skf = StratifiedKFold(n_splits=2)
for train_index, test_index in skf.split(X, y):
   print("TRAIN:", train_index, "TEST:", test_index)
   X_train, X_test = X[train_index], X[test_index]
   y_train, y_test = y[train_index], y[test_index]
```



All mentioned above about k-Fold CV is true for Stratified k-Fold technique. When choosing between different CV methods, make sure you are using the proper one. For example, you might think that your model performs badly simply because you are using k-Fold CV to validate the model which was trained on the dataset with a class imbalance. To avoid that you should always do a proper exploratory data analysis on your data.

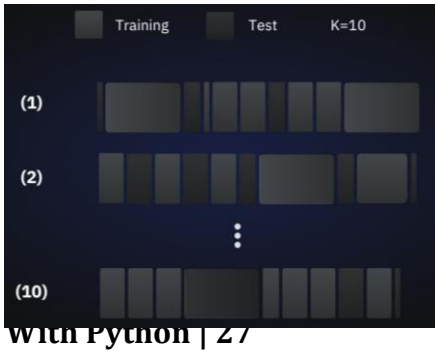
Repeated k-Fold cross-validation

Repeated k-Fold cross-validation or Repeated random sub-sampling CV is probably the most robust of all CV techniques in this paper. It is a variation of k-Fold but in the case of Repeated k-Folds k is not the number of folds. It is the number of times we will train the model.

The general idea is that on every iteration we will randomly select samples all over the dataset as our test set. For example, if we decide that 20% of the dataset will be our test set, 20% of samples will be randomly selected and the rest 80% will become the training set.

The algorithm of Repeated k-Fold technique:

1. Pick k – number of times the model will be trained
2. Pick a number of samples which will be the test set
3. Split the dataset
4. Train on the training set. On each iteration of cross-validation, a new model must be trained
5. Validate on the test set



6. Save the result of the validation
7. Repeat steps 3-6 k times
8. To get the final score average the results that you got on step 6.

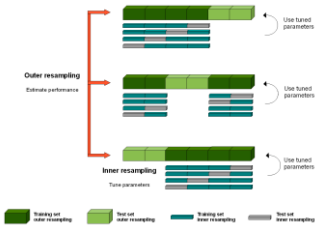
Repeated k-Fold has clear advantages over standard k-Fold CV. Firstly, the proportion of train/test split is not dependent on the number of iterations. Secondly, we can even set unique proportions for every iteration. Thirdly, random selection of samples from the dataset makes Repeated k-Fold even more robust to selection bias.

Still, there are some disadvantages. k-Fold CV guarantees that the model will be tested on all samples, whereas Repeated k-Fold is based on randomization which means that some samples may never be selected to be in the test set at all. At the same time, some samples might be selected multiple times. Thus making it a bad choice for imbalanced datasets.

Sklearn will help you to implement a Repeated k-Fold CV. Just use sklearn.model_selection.RepeatedKFold. In sklearn implementation of this technique you must set the number of folds that you want to have (n_splits) and the number of times the split will be performed (n_repeats). It guarantees that you will have different folds on each iteration.

```
import numpy as np
from sklearn.model_selection import RepeatedKFold
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 1, 1])
rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=42)
for train_index, test_index in rkf.split(X):
  print("TRAIN:", train_index, "TEST:", test_index)
  X_train, X_test = X[train_index], X[test_index]
  y_train, y_test = y[train_index], y[test_index]
```

Nested k-Fold



Unlike the other CV techniques, which are designed to evaluate the quality of an algorithm, Nested k-fold CV is used to train a model in which hyperparameters also need to be optimized. It estimates the generalization error of the underlying model and its (hyper)parameter search.

The algorithm of Nested k-Fold technique:

1. Define set of hyper-parameter combinations, C, for current model. If model has no hyper-parameters, C is the empty set.
2. Divide data into K folds with approximately equal distribution of cases and controls.
3. (outer loop) For fold k, in the K folds:
   - Set fold k, as the test set.
   - Perform automated feature selection on the remaining K-1 folds.
   - For parameter combination c in C:
     - (inner loop) For fold k, in the remaining K-1 folds:
     - Set fold k, as the validation set.
     - Train model on remaining K-2 folds.
     - Evaluate model performance on fold k.
   - Calculate average performance over K-2 folds for parameter combination c.

- Train model on K-1 folds using hyper-parameter combination that yielded best average performance over all steps of the inner loop.
- Evaluate model performance on fold k.

4. Calculate average performance over K folds.

The inner loop performs cross-validation to identify the best features and model hyper-parameters using the k-1 data folds available at each iteration of the outer loop. The model is trained once for each outer loop step and evaluated on the held-out data fold. This process yields k evaluations of the model performance, one for each data fold, and allows the model to be tested on every sample.

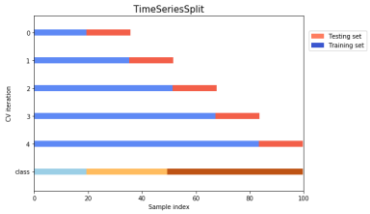
It is to be noted that this technique is computationally expensive because plenty of models is trained and evaluated. Unfortunately, there is no built-in method in sklearn that would perform Nested k-Fold CV for you.

Time-series cross-validation

Traditional cross-validation techniques don't work on sequential data such as time-series because we cannot choose random data points and assign them to either the test set or the train set as it makes no sense to use the values from the future to forecast values in the past. There are mainly two ways to go about this:

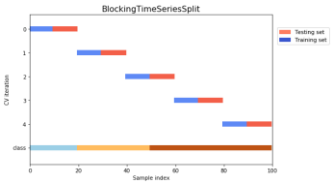
1. Rolling cross-validation

Cross-validation is done on a rolling basis i.e. starting with a small subset of data for training purposes, predicting the future values, and then checking the accuracy on the forecasted data points. The following image can help you get the intuition behind this approach.



2. Blocked cross-validation

The first technique may introduce leakage from future data to the model. The model will observe future patterns to forecast and try to memorize them. That's why blocked cross-validation was introduced.



It works by adding margins at two positions. The first is between the training and validation folds in order to prevent the model from observing lag values which are used twice, once as a repressor and another as a response. The second is between the folds used at each iteration in order to prevent the model from memorizing patterns from one iteration to the next.

Advantages:

1. Overcoming Overfitting: Cross validation helps to prevent overfitting by providing a more robust estimate of the model's performance on unseen data.
2. Model Selection: Cross validation can be used to compare different models and select the one that performs the best on average.
3. Hyper parameter tuning: Cross validation can be used to optimize the hyper parameters of a model, such as the regularization parameter, by selecting the values that result in the best performance on the validation set.
4. Data Efficient: Cross validation allows the use of all the available data for both training and validation, making it a more data-efficient method compared to traditional validation techniques.

Disadvantages:

1. Computationally Expensive: Cross validation can be computationally expensive, especially when the number of folds is large or when the model is complex and requires a long time to train.
2. Time-Consuming: Cross validation can be time-consuming, especially when there are many hyper parameters to tune or when multiple models need to be compared.
3. Bias-Variance Trade-off: The choice of the number of folds in cross validation can impact the bias-variance trade-off, i.e., too few folds may result in high variance, while too many folds may result in high bias.

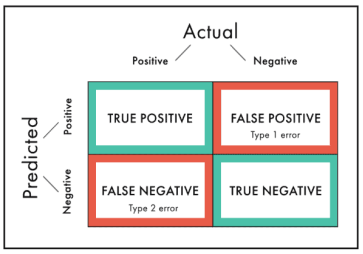
What is a Confusion Matrix?

A **confusion matrix** is a matrix that summarizes the performance of a machine learning model on a set of test data. It is a means of displaying the number of accurate and inaccurate instances based on the model's predictions. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance.

The matrix displays the number of instances produced by the model on the test data.

- **True Positive (TP):** The model correctly predicted a positive outcome (the actual outcome was positive).
- **True Negative (TN):** The model correctly predicted a negative outcome (the actual outcome was negative).
- **False Positive (FP):** The model incorrectly predicted a positive outcome (the actual outcome was negative). Also known as a Type I error.
- **False Negative (FN):** The model incorrectly predicted a negative outcome (the actual outcome was positive). Also known as a Type II error.

The Confusion Matrix



Metrics based on Confusion Matrix Data

1. Accuracy: - Accuracy is used to measure the performance of the model. It is the ratio of Total correct instances to the total instances.

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

For the above case: Accuracy = (5+3)/ (5+3+1+1) = 8/10 = 0.8

2. Precision: - Precision is a measure of how accurate a model's positive predictions are. It is defined as the ratio of true positive predictions to the total number of positive predictions made by the model.

$$Precision = \frac{TP}{(TP + FP)}$$

For the above case: Precision = 5/ (5+1) =5/6 = 0.8333

3. Recall: - Recall measures the effectiveness of a classification model in identifying all relevant instances from a dataset. It is the ratio of the number of true positive (TP) instances to the sum of true positive and false negative (FN) instances.

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

For the above case: Recall = 5/ (5+1) =5/6 = 0.8333

4. F1-Score: - F1-score is used to evaluate the overall performance of a classification model. It is the harmonic mean of precision and recall,

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

Where a β value less than 1 would lower the impact of Precision and vice-versa. At β=1, F-beta becomes F1.

Now that the metrics of a classification problem are under our belt. Let's pick a dataset, train a model and evaluate its performance using a confusion matrix.

For the above case: F1-Score: = (2* 0.8333* 0.8333)/ (0.8333+ 0.8333) = 0.8333

We balance precision and recall with the F1-score when a trade-off between minimizing false positives and false negatives is necessary, such as in information retrieval systems.

5. Specificity: - Specificity is another important metric in the evaluation of classification models, particularly in binary classification. It measures the ability of a model to correctly identify negative instances. Specificity is also known as the True Negative Rate. Formula is given by:

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

For example, Specificity=3/ (1+3) =3/4=0.75

6. Type 1 and Type 2 error

1. Type 1 error

Type 1 error occurs when the model predicts a positive instance, but it is actually negative. Precision is affected by false positives, as it is the ratio of true positives to the sum of true positives and false positives.

$$\text{Type 1 Error} = \frac{FP}{TN+FP}$$

For example, in a courtroom scenario, a Type 1 Error, often referred to as a false positive, occurs when the court mistakenly convicts an individual as guilty when, in truth, they are innocent of the alleged crime. This grave error can have profound consequences, leading to the wrongful punishment of an innocent person who did not commit the offense in question. Preventing Type 1 Errors in legal proceedings is paramount to ensuring that justice is accurately served and innocent individuals are protected from unwarranted harm and punishment.

2. Type 2 error

Type 2 error occurs when the model fails to predict a positive instance. Recall is directly affected by false negatives, as it is the ratio of true positives to the sum of true positives and false negatives.

$$\text{Type 2 Error} = \frac{FN}{TP+FN}$$

In the context of medical testing, a Type 2 Error, often known as a false negative, occurs when a diagnostic test fails to detect the presence of a disease in a patient who genuinely has it. The consequences of such an error are significant, as it may result in a delayed diagnosis and subsequent treatment.

Precision emphasizes minimizing false positives, while recall focuses on minimizing false negatives.

Support Vector Machine (SVM) Algorithm

A Support Vector Machine (SVM) is a powerful machine learning algorithm widely used for both linear and nonlinear classification, as well as regression and outlier detection tasks. SVMs are highly adaptable, making them suitable for various applications such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection and anomaly detection.

SVMs are particularly effective because they focus on finding the maximum separating hyperplane between the different classes in the target feature, making them robust for both binary and multiclass classification. In this outline, we will explore the Support Vector Machine (SVM) algorithm, its applications, and how it effectively

handles both linear and nonlinear classification, as well as regression and outlier detection tasks.

Support Vector Machine

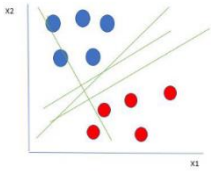
A **Support Vector Machine (SVM)** is a supervised machine learning **algorithm** used for both **classification** and **regression** tasks. While it can be applied to regression problems, SVM is best suited for **classification** tasks. The primary objective of the **SVM algorithm** is to identify the **optimal hyperplane** in an N-dimensional space that can effectively separate data points into different classes in the feature space. The algorithm ensures that the margin between the closest points of different classes, known as **support vectors**, is maximized.

The dimension of the hyperplane depends on the number of features. For instance, if there are two input features, the hyperplane is simply a line, and if there are three input features, the hyperplane becomes a 2-D plane. As the number of features increases beyond three, the complexity of visualizing the hyperplane also increases.

Consider two independent variables, **x1** and **x2**, and one dependent variable represented as either a blue circle or a red circle.

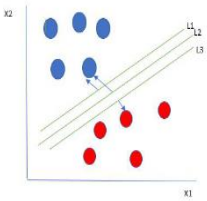
- In this scenario, the hyperplane is a line because we are working with two features (**x1** and **x2**).
- There are multiple lines (or **hyperplanes**) that can separate the data points.
- The challenge is to determine the **best hyperplane** that maximizes the separation margin between the red and blue circles.

Linearly Separable Data points



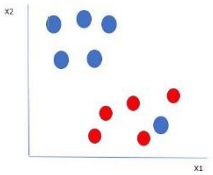
From the figure above it's very clear that there are multiple lines (our hyperplane here is a line because we are considering only two input features x1, x2) that segregate our data points or do a classification between red and blue circles. So how do we choose the best line or in general the best hyperplane that segregates our data points?

How does Support Vector Machine Algorithm Work?



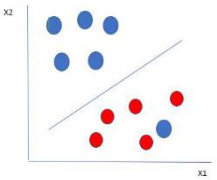
One reasonable choice for the **best hyperplane** in a **Support Vector Machine (SVM)** is the one that maximizes the separation margin between the two classes. The maximum-margin hyperplane, also referred to as the hard margin, is selected based on maximizing the distance between the hyperplane and the nearest data point on each side.

Multiple hyperplanes separate the data from two classes



So we choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the **maximum-margin hyperplane/hard margin**. So from the above figure, we choose L2. Let's consider a scenario like shown below

Selecting hyperplane for data with outlier



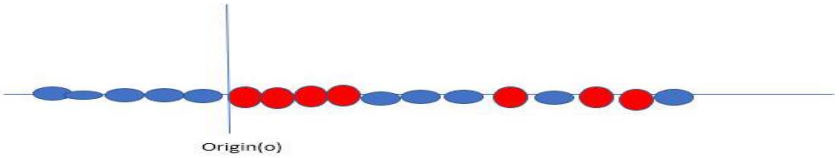
Here we have one blue ball in the boundary of the red ball. So how does SVM classify the data? It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics

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

So in this type of data point what SVM does is, finds the maximum margin as done with previous data sets along with that it adds a penalty each time a point crosses the margin. So the margins in these types of cases are called **soft margins**. When there is a soft margin to the data set, the SVM tries to minimize *(1/margin+(∑penalty))*. Hinge loss is a commonly used penalty. If no violations no hinge loss. If violations hinge loss proportional to the distance of 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?



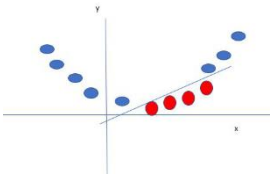
Original 1D dataset for classification

Say, our data is shown in the figure above. SVM solves this by creating a new variable using a **kernel**. We call a point x_i on the line and we create a new variable y_i as a function of distance from origin o.so if we plot this we get something like as shown below

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. A non-linear function that creates a new variable is referred to as a kernel.

Support Vector Machine Terminology



- **Hyperplane**: The **hyperplane** is the decision boundary used to separate data points of different classes in a feature space. For **linear classification**, this is a linear equation represented as wx+b=0.
- **Support Vectors**: **Support vectors** are the closest data points to the hyperplane. These points are critical in determining the hyperplane and the margin in **Support Vector Machine (SVM)**.
- **Margin**: The **margin** refers to the distance between the **support vector** and the hyperplane. The primary goal of the SVM algorithm is to maximize this margin, as a wider margin typically results in better classification performance.
- **Kernel**: The **kernel** is a mathematical function used in SVM to map input data into a higher-dimensional feature space. This allows the SVM to find a hyperplane in cases where data points are not linearly separable in the original space. Common **kernel functions** include linear, polynomial, radial basis function (RBF), and sigmoid.
- **Hard Margin**: A **hard margin** refers to the maximum-margin hyperplane that perfectly separates the data points of different classes without any misclassifications.
- **Soft Margin**: When data contains **outliers** or is not perfectly separable, SVM uses the **soft margin** technique. This method introduces a **slack variable** for each data point to allow some misclassifications while balancing between maximizing the margin and minimizing violations.

- **C**: The **C parameter** in SVM is a regularization term that balances margin maximization and the penalty for misclassifications. A higher **C** value imposes a stricter penalty for margin violations, leading to a smaller margin but fewer misclassifications.
- **Hinge Loss**: The **hinge loss** is a common loss function in SVMs. It penalizes misclassified points or margin violations and is often combined with a regularization term in the objective function.
- **Dual Problem**: The **dual problem** in SVM involves solving for the **Lagrange multipliers** associated with the support vectors. This formulation allows for the use of the **kernel trick** and facilitates more efficient computation.

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 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. 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.

Popular kernel functions in SVM

The SVM kernel is a function that takes low-dimensional input space and transforms it into higher-dimensional space, i.e. it converts non separable problems to separable problems. It is mostly useful in non-linear separation problems. Simply put the kernel, does some extremely complex data transformations and then finds out the process to separate the data based on the labels or outputs defined.

Linear: $K(w,b)=w^T x+b$

Polynomial: $K(w,x)=(\gamma w^T x+b)^N$

Gaussian RBF: $K(w,x)=\exp(-\gamma||x_i-x_j||^n)$

Sigmoid: $K(xi,xj)=\tanh(\alpha x_i^T x_j+b)$

Implementing SVM Algorithm in Python

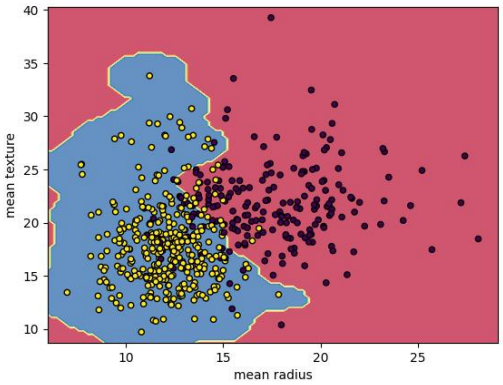
Predict if cancer is Benign or malignant. Using historical data about patients diagnosed with cancer enables doctors to differentiate malignant cases and benign ones are given independent attributes.

Steps

- 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.
- Plot the decision boundary.
- Plot the decision boundary

 # 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()
```

Advantages

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

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.

Extracting Confidence Measurements

In predictive modelling, confidence measurement refers to quantifying how certain or uncertain the model is about its predictions. It helps assess the reliability of the model's output and can be valuable in decision-making, especially in uncertain or risky contexts. The confidence can be interpreted differently depending on the type of model used (e.g., classification, regression) and the underlying algorithm (e.g., decision trees, neural networks).

Here are different approaches to extracting confidence measures from predictive models:

1. **Confidence in Classification Models**

In classification tasks, confidence is typically measured by how confident the model is about its predicted class. For example, if the model predicts a class, it often provides a probability or score associated with that prediction.

Methods for Measuring Confidence in Classification:

- Probabilistic Output (e.g., Logistic Regression, Naive Bayes): Many classification algorithms (like Logistic Regression or Naive Bayes) output a probability score that represents the likelihood that a data point belongs to a particular class. This probability can be interpreted as the model's confidence in that prediction.

  For example, in a binary classification problem, If the model predicts "class 1" with a probability of 0.85, it means the model is 85% confident that the sample belongs to class 1. Example (using Logistic Regression with scikit-learn):

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Assuming X, y are the features and labels of the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
 # Train a logistic regression model
model = LogisticRegression() model.fit(X_train, y_train)
 # Predict probabilities (confidence scores)
y_probs = model.predict_proba(X_test)
# The first column corresponds to the probability of class 0, and the second to
class 1 print(y_probs[:5]) # Print the first 5 probability values
```

  The output y_probs will be a matrix where each row contains two values: the probability of each class. For example, [0.1, 0.9] means the model is 10% confident that the sample belongs to class 0 and 90% confident that it belongs to class 1.
- **Support Vector Machines (SVMs):** SVMs with probability estimation (using Platt scaling) can also return class probabilities that indicate confidence levels. These probabilities are derived from the distance of the point to the decision boundary.
- **Random Forests and Gradient Boosting** (e.g., XGBoost, LightGBM): Random Forests and other ensemble models can provide probabilities for each class, which can be averaged over all the trees in the forest. These probabilities can be interpreted as confidence scores.

Example (Random Forest Classifier):

```
from sklearn.ensemble import RandomForestClassifier
# Train a random forest classifier
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train) # Get class probabilities
y_probs = model.predict_proba(X_test)
print(y_probs[:5]) # Print the first 5 probability values
```

This gives a similar result: the model's "confidence" in the predicted class as a probability.

- **Threshold for Confidence:**

Sometimes, a threshold is set to decide whether the model's confidence is high enough to make a decision. For example, if the model predicts class 1 with a probability greater than 0.7, you might classify the sample as class 1; otherwise, you might classify it as "uncertain" or require further analysis.

2. **Confidence in Regression Models**

In regression problems, confidence is often measured by the uncertainty or the range within which the true value might lie. This is typically provided by methods like confidence intervals, prediction intervals, or variance estimates.

Methods for Measuring Confidence in Regression:

- **Prediction Interval:** A prediction interval is a range around the predicted value within which you expect the true output value to fall, with a certain level of confidence (e.g., 95%). This gives a measure of how uncertain the model's prediction is. For example, if the model predicts a value of 50, and the prediction interval is [45, 55], it means the model is 95% confident that the true value lies within this range.
- **Standard Error of Prediction:** In linear regression, the model can provide a standard error of the predictions. The lower the standard error, the more confident the model is in its predictions.
- **Quantile Regression**: For a regression problem, quantile regression can be used to estimate the conditional quantiles of the target variable. This helps provide a range of possible outcomes (e.g., the 90th percentile prediction), rather than a single point prediction.

Example for Confidence Interval in Regression:

Using statsmodels in Python, we can obtain confidence intervals for the predictions:

```
import statsmodels.api as sm
import numpy as np
# Assume X_train, X_test, and y_train are the features and labels X_train_with_intercept =
sm.add_constant(X_train) # Add constant for intercept
model = sm.OLS(y_train, X_train_with_intercept)
results = model.fit() # Make predictions with confidence intervals X_test_with_intercept =
sm.add_constant(X_test)
predictions = results.get_prediction(X_test_with_intercept)
summary_frame = predictions.summary_frame(alpha=0.05) # 95% confidence interval
print(summary_frame[['mean', 'obs_ci_lower', 'obs_ci_upper']])
```

The summary_frame will give you the predicted values, along with lower and upper bounds of the confidence intervals (e.g., for 95% confidence).

3. Uncertainty Quantification in Deep Learning:
For deep learning models like neural networks, confidence can be measured using techniques such as:

- Monte Carlo Dropout: Applying dropout during inference (instead of only during training) to approximate a Bayesian posterior distribution over predictions, which gives an estimate of uncertainty.
- Ensemble Methods: Using an ensemble of models to generate different predictions and assess the spread (variance) across them to measure uncertainty.

Code Example: Monte Carlo Dropout (using Keras):

```
import keras
import numpy as np # Assuming `model` is a trained neural network
def predict_with_uncertainty(model, X, n_iter=100):
        f = keras.backend.function([model.input], [model.output])
        predictions = np.array([f([X])[0] for _ in range(n_iter)])
        mean_predictions = predictions.mean(axis=0)
        uncertainty = predictions.std(axis=0)
        return mean_predictions, uncertainty
# Predict with uncertainty
mean_preds, uncertainty = predict_with_uncertainty(model, X_test)
```

In this case, the uncertainty represents how much the predictions vary across the different stochastic forward passes (with dropout active), which reflects the model's confidence.

4. Bayesian Methods:

Bayesian models, such as Bayesian Linear Regression or Gaussian Processes, naturally provide uncertainty estimates with each prediction. These models treat predictions probabilistically, and they give a distribution over possible outcomes rather than a single deterministic output.

Unit -3 Unsupervised Learning

Unsupervised Machine Learning

we learned supervised machine learning in which models are trained using labeled data under the supervision of training data. But there may be many cases in which we do not have labeled data and need to find the hidden patterns from the given dataset. So, to solve such types of cases in machine learning, we need unsupervised learning techniques.

What is Unsupervised Learning?

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format**.

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

Why use Unsupervised Learning?

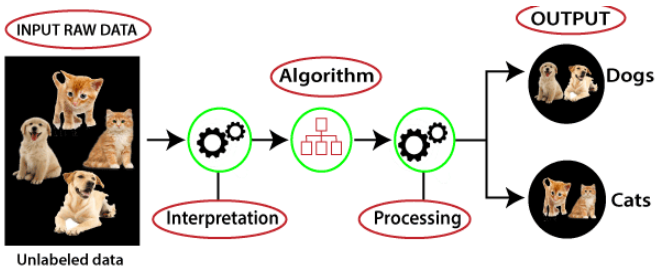
Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of Unsupervised Learning

Working of unsupervised learning can be understood by the below diagram:

Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning



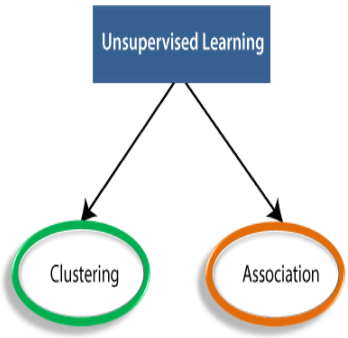
model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:

- **Clustering**: Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.
- **Association**: An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.



Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

- **K-means clustering**
- **KNN (k-nearest neighbors)**
- **Hierarchal clustering**
- **Anomaly detection**
- **Neural Networks**
- **Principle Component Analysis**
- **Independent Component Analysis**
- **Apriori algorithm**
- **Singular value decomposition**

Advantages

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

K-means clustering

K-means clustering, originating from signal processing, is a technique in vector quantization. Its objective is to divide a set of n observations into k clusters, with each observation assigned to the cluster whose mean (cluster center or centroid) is closest, thereby acting as a representative of that cluster. K-means clustering, a part of the unsupervised learning family in AI, is used to group similar data points together in a process known as clustering. Clustering helps us understand our data in a unique way - by grouping things together into- you guessed it - clusters.

What is K-Means Clustering?

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into a pre-defined number of clusters. The goal is to group similar data points together and discover underlying patterns or structures within the data.

Recall the first property of clusters – it states that the points within a cluster should be similar to each other. So, **our aim here is to minimize the distance between the points within a cluster.**

There is an algorithm that tries to minimize the distance of the points in a cluster with their centroid – the k-means clustering technique.

K-means is a centroid-based algorithm or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid.

The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.

Optimization plays a crucial role in the k-means clustering algorithm. The goal of the optimization process is to find the best set of centroids that minimizes the sum of squared distances between each data point and its closest centroid.

How K-Means Clustering Works?

Here's how it works:

1. **Initialization**: Start by randomly selecting K points from the dataset. These points will act as the initial cluster centroids.
2. **Assignment**: For each data point in the dataset, calculate the distance between that point and each of the K centroids. Assign the data point to the cluster whose centroid is closest to it. This step effectively forms K clusters.
3. **Update centroids**: Once all data points have been assigned to clusters, recalculate the centroids of the clusters by taking the mean of all data points assigned to each cluster.
4. **Repeat**: Repeat steps 2 and 3 until convergence. Convergence occurs when the centroids no longer change significantly or when a specified number of iterations is reached.
5. **Final Result**: Once convergence is achieved, the algorithm outputs the final cluster centroids and the assignment of each data point to a cluster.

Objective of k means Clustering

The main objective of k-means clustering is to partition your data into a specific number (k) of groups, where data points within each group are similar and dissimilar to points in other groups. It achieves this by minimizing the distance between data points and their assigned cluster's center, called the centroid.

Here's an objective:

- **Grouping similar data points:** K-means aims to identify patterns in your data by grouping data points that share similar characteristics together. This allows you to discover underlying structures within the data.
- **Minimizing within-cluster distance:** The algorithm strives to make sure data points within a cluster are as close as possible to each other, as measured by a distance metric (usually Euclidean distance). This ensures tight-knit clusters with high cohesiveness.
- **Maximizing between-cluster distance:** Conversely, k-means also tries to maximize the separation between clusters. Ideally, data points from different clusters should be far apart, making the clusters distinct from each other.

Working of K-Means Algorithm

The following stages will help us understand how the K-Means clustering technique works-

- ***Step 1:*** First, we need to provide the number of clusters k , that need to be generated by this algorithm.
- ***Step 2:*** Next, choose K data points at random and assign each to a cluster. Briefly, categorize the data based on the number of data points.
- ***Step 3:*** The computation of initial cluster centroids will now be performed.
- ***Step 4:*** Iterate the steps below until we find the ideal centroid, which is the assigning of data points to the closest cluster centroids, ensuring minimal variance within each cluster.
    - 4.1 The sum of squared distances between data points and initial centroids would be calculated first.
    - 4.2 At this point, we need to allocate each data point to the cluster that is closest to the others nearest centroid.
    - 4.3 Finally, compute the closest centroids for the clusters by averaging all of the cluster's data points.

K-means implements the **Expectation-Maximization** strategy to solve the problem. The Expectation-step is used to assign data points to the nearest cluster, and the Maximization-step is used to compute the centroid of each cluster.

Following points using the K-means algorithm

- It is suggested to normalize the data while dealing with clustering algorithms such as K-Means since such algorithms employ distance-based measurement to identify the similarity between data points.
- Because of the iterative nature of K-Means and the random initialization of centroids, K-Means may become stuck in a local optimum and fail to converge to the global optimum. As a result, it is advised to employ distinct centroids' initializations.

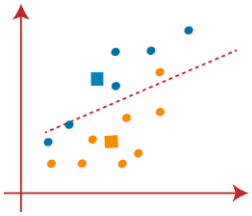
Implementation of K Means Clustering Graphical Form

Step 1: Let us pick k clusters, i.e., K=2, to separate the dataset and assign it to its appropriate clusters. We will select two random places to function as the cluster's centroid.

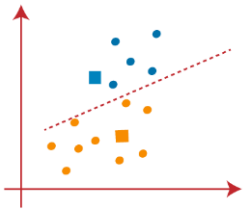
Step 2: Now, each data point will be assigned to a scatter plot depending on its distance from the nearest K-point or centroid. This will be accomplished by establishing a median between both centroids. Consider the following illustration:

Step 3: The points on the line's left side are close to the blue centroid, while the points on the line's right side are close to the yellow centroid. The left Form cluster has a blue centroid, whereas the right Form cluster has a yellow centroid.

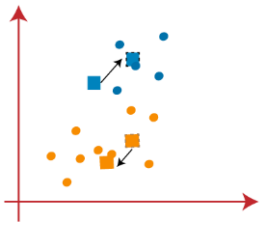
Step 4: Repeat the procedure, this time selecting a different centroid. To choose the new centroids, we will determine their new center of gravity, which is represented below.



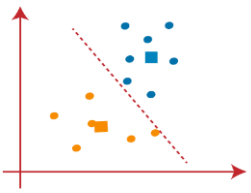
Step 5: After that, we'll re-assign each data point to its new centroid. We shall repeat the procedure outlined before (using a median line). The blue cluster will contain the yellow data point on the blue side of the median line.



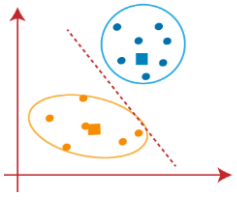
Step 6: Now that reassignment has occurred, we will repeat the previous step of locating new centroids.



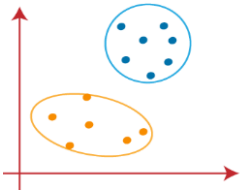
Step 7: We will repeat the procedure outlined above for determining the center of gravity of centroids, as shown below.



Step 8: Similar to the previous stages, we will draw the median line and reassign the data points after locating the new centroids.



Step 9: We will finally group points depending on their distance from the median line, ensuring that two distinct groups are established and that no dissimilar points are included in a single group.



The final Cluster is as follows:

Advantages

- It is simple to grasp and put into practice.
- K-means would be faster than Hierarchical clustering if we had a high number of variables.
- An instance's cluster can be changed when centroids are re-computation.
- When compared to Hierarchical clustering, K-means produces tighter clusters.

Disadvantages

- The number of clusters, i.e., the k value, is difficult to estimate.
- A major effect on output is exerted by initial inputs such as the number of clusters in a network (k value).
- The sequence in which the data is entered has a considerable impact on the final output.
- It's quite sensitive to rescaling. If we rescale our data using normalization or standards, the outcome will be drastically different. ultimate result
- It is not advisable to do clustering tasks if different clusters have a sophisticated geometric shape.

Vector quantization

Vector quantization (VQ) is a technique used for compressing images by representing groups of pixels (vectors) with a single value (codebook entry). Here's a simplified overview of how vector quantization can be applied to compress images:

1. **Dividing the Image into Blocks:**

   The image is divided into small blocks of pixels. Typically, these blocks are squares of fixed size, such as 8x8 pixels.

2. **Building a Codebook:**

   A codebook is created which contains a set of representative vectors (code words). These vectors are usually derived from the image data itself or through a clustering algorithm like k-means clustering.

3. **Encoding:**

   Each block of pixels in the image is compared to the code words in the codebook. The closest codeword (in terms of some distance metric, often Euclidean distance) is chosen to represent that block of pixels.

4. **Storing the Codewords:**

   Instead of storing the entire image, we store the indices of the code words from the codebook that best represent each block of pixels.

   The size of the codebook directly affects the compression ratio and image quality. A larger codebook provides better quality but requires more storage.

5. **Decoding:**

   To reconstruct the image, each index in the compressed data is replaced with the corresponding codeword from the codebook. This process recreates the image from the compressed representation.

vector quantization provides a powerful method for compressing images by representing groups of pixels with a smaller set of representative vectors. This compression method balances between storage efficiency and image quality, making it suitable for various applications where image data needs to be transmitted or stored efficiently.

Below is an example of compressing an image using vector quantization in Python. We'll use the popular scikit-learn library for k-means clustering, which can be considered a form of vector quantization for this purpose.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error
from PIL import Image

# Function to compress an image using vector quantization
def compress_image(image, num_colors=16):
    # Convert image to numpy array
    image_array = np.array(image, dtype=np.float64) / 255

    # Reshape the image to 2D array of pixels (height * width, channels)
    original_shape = image_array.shape
    image_array = image_array.reshape((-1, 3))

    # Shuffle the pixels to get a more accurate clustering
    image_array_sample = shuffle(image_array, random_state=0)[:1000]

    # Perform k-means clustering
    kmeans = KMeans(n_clusters=num_colors, random_state=0)
    kmeans.fit(image_array_sample)

    # Predict color indices for all pixels
    labels = kmeans.predict(image_array)

    # Create the compressed image
    compressed_image = np.zeros_like(image_array)
    for i in range(num_colors):
        compressed_image[labels == i] = kmeans.cluster_centers_[i]

    # Reshape the compressed image back to the original shape
    compressed_image = compressed_image.reshape(original_shape)

    # Convert back to uint8 and return as Image object
    compressed_image = (compressed_image * 255).astype(np.uint8)
    return Image.fromarray(compressed_image)

# Load an example image
image_path = 'example_image.jpg'
original_image = Image.open(image_path)

# Display the original image
plt.figure(figsize=(6, 6))
```

```
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(original_image)
plt.axis('off')
# Compress the image using vector quantization (k-means)
compressed_image = compress_image(original_image)
compressed_image.save('compressed_image.jpg')  # Save the compressed image

# Display the compressed image
plt.subplot(1, 2, 2)
plt.title('Compressed Image')
plt.imshow(compressed_image)
plt.axis('off')

plt.show()
```

Explanation:

1. **Imports and Setup:**
   We import necessary libraries such as numpy, matplotlib (for plotting), scikit-learn (for k-means clustering), and PIL (Python Imaging Library) for image handling.
2. **compress_image Function:**
   This function takes an image as input and compresses it using vector quantization (k-means clustering). It converts the image to a numpy array and reshapes it into a 2D array of pixels. It performs k-means clustering on a sample of pixels to find num_colors centroids (colors). It assigns each pixel in the image to the closest centroid (cluster) and reconstructs the compressed image using these centroids. Finally, it reshapes the compressed image back to its original shape and converts it to an Image object before returning.
3. **Loading and Displaying the Original Image:**
   We load an example image (example_image.jpg) using Image.open() from PIL and display it using matplotlib.
4. **Compressing the Image:**
   We call compress_image() to compress the loaded image using vector quantization (k-means clustering).
5. **Saving and Displaying the Compressed Image:**
   The compressed image is saved to compressed_image.jpg and displayed alongside the original image using matplotlib.

Advantages :

- **Lossy Compression:** It achieves compression by sacrificing some image quality, but the trade-off can be controlled by the size of the codebook.
- **Fast Decoding:** Once the codebook is constructed, decoding (reconstructing the image) is relatively fast.
- **Adaptive to Image Content:** The codebook can adapt to the statistics of the image, making it efficient for different types of images.

Challenges:

- **Codebook Design:** Constructing an optimal codebook that minimizes the overall distortion while maintaining a reasonable size can be challenging.

- **High Complexity for Large Codebooks:** Managing a large codebook requires significant computational resources and storage.

Applications:

- Vector quantization has been widely used in image and video compression standards like JPEG (Joint Photographic Experts Group) and MPEG (Moving Picture Experts Group).

Means Shift Clustering

Clustering is a fundamental method in unsupervised device learning, and one powerful set of rules for this venture is Mean Shift clustering. Mean Shift is a technique for grouping comparable data factors into clusters primarily based on their inherent characteristics.

Mean Shift clustering, together with using the scikit-study library in Python to use this method. We'll cover key concepts like clustering, Kernel Density Estimation (KDE), and bandwidth, and offer step-by-step commands for acting Mean Shift clustering with the usage of scikit-analyze.

Mean-shift algorithm basically assigns the data-points to the clusters iteratively by shifting points towards the highest density of data-points i.e. cluster centroid.

Mean Shift is a clustering algorithm used to pick out dense areas in a dataset and assign facts and factors to their respective clusters. It is a non-parametric, density-based clustering technique, which means it does not require any previous assumptions approximately the wide variety of clusters or their shapes. Instead, it discovers clusters based totally on the density of information points within the function area.

Mean Shift is a mode-seeking algorithm, which means that it finds the modes (peaks) of the density distribution of the data. This is in contrast to centroid-based clustering algorithms, such as K-Means clustering, which find the centroids of the clusters.

Mean Shift works by iteratively refining the positions of the data points, moving them towards the modes of the density distribution. This process is repeated until the data points converge to the modes of the density distribution.

Algorithm

The Mean Shift Algorithm follows a specific sequence of operations to move statistics factors closer to the modes of the anticipated density.

1. **Initialization:** Start with hard and fast data points.
2. **Bandwidth Selection:** Choose the ideal bandwidth h for the kernel. You can use techniques like Scott's Rule or Silverman's Rule or estimate it using move-validation.
3. **Mean Shift Computation:** For each data point, compute the mean shift vector M(x).
4. **Shift Data Points:** Update the data points by shifting them in the direction of M(x). x←x+M(x).
5. **Convergence Check:** Repeat steps three and four until the data points do not converge. You can outline a convergence threshold.
6. **Cluster Assignment:** Once the data factors have converged, assign every point to its nearest mode (cluster middle).
7. **Result:** The final clusters are determined based on the modes.

Mean-Shift Clustering Work

1. **Kernel Density Estimation:** The first step involves estimating the underlying probability density function (PDF) of the data points. This is typically done using kernel density estimation, where each data point is represented by a kernel function cantered at that point. The kernel function specifies the weight assigned to each data point in the density estimation process.
2. **Shifting Data Points:** In the second step, the algorithm iteratively shifts the data points towards regions of higher density. The shift is determined by calculating the mean shift vector for each data point, which represents the direction and magnitude of the shift. The mean shift vector is calculated as the weighted average of the differences between the data point and its neighbouring points, where the weights are determined by the kernel function.
3. **Convergence and Cluster Identification:** The algorithm continues shifting the data points until convergence is reached. Convergence occurs when the mean shift vectors become very small or negligible. Once convergence is achieved, the final position of each data point represents a cluster center. The algorithm assigns each data point to the closest cluster center, thereby identifying the clusters within the data.

Applications of Mean-Shift Clustering

The Mean-Shift clustering algorithm has several applications in various fields.

- Computer vision − Mean-Shift clustering is widely used in computer vision for object tracking, image segmentation, and feature extraction.
- Image processing − Mean-Shift clustering is used for image segmentation, which is the process of dividing an image into multiple segments based on the similarity of the pixels.
- Anomaly detection − Mean-Shift clustering can be used for detecting anomalies in data by identifying the areas with low density.
- Customer segmentation − Mean-Shift clustering can be used for customer segmentation in marketing by identifying groups of customers with similar behavior and preferences.
- Social network analysis − Mean-Shift clustering can be used for clustering users in social networks based on their interests and interactions.

Mean removal is a preprocessing step often used in data analysis and machine learning to center the data around zero. This process involves subtracting the mean of each feature from the data points.

Here's a simple example using NumPy and pandas:

Example using NumPy	**Example using pandas**

import numpy as np
# Example data: 2D array (features x samples) 	
data = np.array([[1.0, 2.0, 3.0],
[4.0, 5.0, 6.0],
[7.0, 8.0, 9.0]])
# Compute the mean of each feature (column-wise mean) 	
mean = np.mean(data, axis=0)
	import pandas as pd
	# Example data in a pandas DataFrame
	data = pd.DataFrame({
	'Feature1': [1.0, 4.0, 7.0],
	'Feature2': [2.0, 5.0, 8.0],
	'Feature3': [3.0, 6.0, 9.0]
	})
	# Compute the mean of each feature
	mean = data.mean()

# Remove the mean from the data 	
mean_removed_data = data - mean
print("Original data:")
print(data)
print("\nMean of each feature:")
print(mean)
print("\nMean removed data:")
print(mean_removed_data)
	# Remove the mean from the data
	mean_removed_data = data - mean
	print("Original data:")
	print(data)
	print("\nMean of each feature:")
	print(mean)
	print("\nMean removed data:")
	print(mean_removed_data)

If your data is in a pandas Data Frame, you can achieve the same result as follows:

Explanation

- **Data**: We have a dataset with features organized in rows and columns.
- **Mean Calculation**: `np.mean(data, axis=0)` calculates the mean of each column (feature). In pandas, `data.mean()` does the same for each column.
- **Mean Removal**: We subtract the mean from the original data to center it around zero.

In both examples, the result is a new dataset where each feature has been centered by removing its mean. This preprocessing step can help in improving the performance of many machine learning algorithms.

Advantages

- It does not need to make any model assumption as like in K-means or Gaussian mixture.
- It can also model the complex clusters which have nonconvex shape.
- It only needs one parameter named bandwidth which automatically determines the number of clusters.
- There is no issue of local minima as like in K-means.
- No problem generated from outliers.

Disadvantages

- Mean-shift algorithm does not work well in case of high dimension, where number of clusters changes abruptly.
- We do not have any direct control on the number of clusters but in some applications, we need a specific number of clusters.
- It cannot differentiate between meaningful and meaningless modes.

Agglomerative clustering

Agglomerative Clustering is a hierarchical clustering technique that builds a hierarchy of clusters in a bottom-up manner. It starts by treating each data point as its own cluster and iteratively merges the closest pairs of clusters until all points belong to a single cluster.

Algorithm

1. **Initialization**:
   Start with each data point as its own individual cluster. If there are **nnn** data points, you initially have **nnn** clusters.
2. **Compute Distance Matrix**:
   Calculate the distance between each pair of clusters. Initially, this distance is simply the distance between individual data points.
3. **Merge Closest Clusters**:

Identify the two closest clusters (or individual points) and merge them into a single cluster. The distance matrix is updated to reflect this merging.

4. **Update Distances**:
   Recomputed the distances between the new cluster and the remaining clusters. The method used to update distances depends on the linkage criterion:
   - **Single Linkage (Minimum Linkage)**: Distance between the closest pair of points in the two clusters.
   - **Complete Linkage (Maximum Linkage)**: Distance between the farthest pair of points in the two clusters.
   - **Average Linkage**: Average distance between all pairs of points in the two clusters.
   - **Ward's Linkage**: Minimizes the total within-cluster variance. This method is based on minimizing the increase in the total variance within all clusters when two clusters are merged.
5. **Repeat**:
   Repeat steps 3 and 4 until all data points are merged into a single cluster or until the desired number of clusters is reached.
6. **Termination**:
   Stop when a single cluster contains all data points or when the specified number of clusters is achieved.
7. **Output**:
   The result is typically represented as a tree-like diagram called a dendrogram, which shows how clusters are merged at different levels of similarity.

Key Points:

- **Hierarchical Nature:** Agglomerative clustering builds a hierarchical decomposition of the dataset.
- **Noisy Data Handling:** It is robust to noisy data because it can handle outliers by forming small clusters first.
- **Time Complexity:** The time complexity can be O(N3) O(N^3) O(N3) in the worst case due to the need to repeatedly update the distance matrix, where NNN is the number of data points. However, efficient implementations and optimizations exist in libraries like **scipy and scikit-learn.**
- **Choice of Linkage:** The choice of linkage criterion (single, complete, average) affects the shape and structure of the resulting clusters. It is important to choose the linkage criterion that best fits the data and the clustering goal.

Step 1: Import Libraries and Generate Data

```
# Import Files
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
# Generate synthetic data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Step 2: Perform Agglomerative Clustering

```
# Define the model
model = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
model.fit(X)                    # Fit the model
labels = model. Labels_         # Get cluster labels
```

Step 3: Visualize the Clusters

```
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title('Agglomerative Clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Step 4: Visualize the Dendrogram

To understand how clusters are merged, you can visualize the dendrogram.

```
# Compute the linkage matrix
Z = linkage(X, 'ward')
# Create the dendrogram
plt.figure(figsize=(10, 7))
dendrogram(Z)
plt.title('Dendrogram')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.show()
```

Output

- **Cluster Visualization**:
  The scatter plot shows the data points coloured by the cluster they belong to. This visualization helps you understand how the algorithm has grouped the data.
- **Dendrogram**:
  The dendrogram illustrates how clusters are merged at each step. The y-axis represents the distance between clusters at which they were merged. The x-axis represents the individual data points or clusters being merged. By looking at the height where the branches merge, you can infer the distance or dissimilarity between clusters.

Advantages

- **Hierarchical Structure**: Provides a hierarchy of clusters which can be visualized with a dendrogram.
- **No Need to Specify Number of Clusters Upfront**: You can cut the dendrogram at different levels to obtain different numbers of clusters.

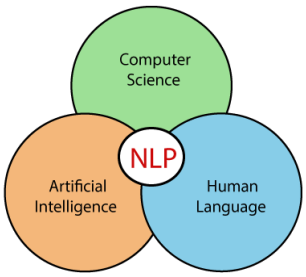
Disadvantages

- **Computational Complexity**: Can be computationally expensive for large datasets.
- **Not Ideal for Large Datasets**: The algorithm may be slow on very large datasets due to its time complexity, which is typically O(n3) O(n^3) O(n3).

Agglomerative clustering is useful for many applications and can be particularly insightful when the hierarchical structure of data is important for analysis.

Unit -4 Natural Language Processing

What is NLP?



NLP stands for **Natural Language Processing**, which is a part of **Computer Science, Human language,** and **Artificial Intelligence**. It is the technology that is used by machines to understand, analyses, manipulate, and interpret human's languages. It helps developers to organize knowledge for performing tasks such as **translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction,** and **topic segmentation**.

Components of NLP

There are the following two components of NLP -

Natural Language Understanding (NLU)

Natural Language Understanding (NLU) helps the machine to understand and analyses human language by extracting the metadata from content such as concepts, entities, keywords, emotion, relations, and semantic roles.

NLU mainly used in Business applications to understand the customer's problem in both spoken and written language. NLU involves the following tasks

- It is used to map the given input into useful representation.
- It is used to analyse different aspects of the language.

Natural Language Generation (NLG)

Natural Language Generation (NLG) acts as a translator that converts the computerized data into natural language representation. It mainly involves Text planning, Sentence planning, and Text Realization.

Difference between NLU and NLG

NLU	NLG
NLU is the process of reading and interpreting language.	NLG is the process of writing or generating language.
It produces non-linguistic outputs from natural language inputs.	It produces constructing natural language outputs from non-linguistic inputs.

NLP Libraries

- **Scikit-learn:** It provides a wide range of algorithms for building machine learning models in Python.
- **Natural language Toolkit (NLTK):** NLTK is a complete toolkit for all NLP techniques.
- **Pattern:** It is a web mining module for NLP and machine learning.
- **TextBlob:** It provides an easy interface to learn basic NLP tasks like sentiment analysis, noun phrase extraction, or pos-tagging.
- **Query:** Query is used to transform natural language questions into queries in a database query language.

- **SpaCy:** SpaCy is an open-source NLP library which is used for Data Extraction, Data Analysis, Sentiment Analysis, and Text Summarization.

Difference between Natural language and Computer Language

Natural Language	Computer Language
Natural language has a very large vocabulary.	Computer language has a very limited vocabulary.
Natural language is easily understood by humans.	Computer language is easily understood by the machines.
Natural language is ambiguous in nature.	Computer language is unambiguous.

Advantages of NLP

- NLP helps users to ask questions about any subject and get a direct response within seconds.
- NLP offers exact answers to the question means it does not offer unnecessary and unwanted information.
- NLP helps computers to communicate with humans in their languages.
- It is very time efficient.
- Most of the companies use NLP to improve the efficiency of documentation processes, accuracy of documentation, and identify the information from large databases.

Disadvantages of NLP

- NLP may not show context.
- NLP is unpredictable
- NLP may require more keystrokes.
- NLP is unable to adapt to the new domain, and it has a limited function that's why NLP is built for a single and specific task only.

Pre-processing Data

Natural Language Processing (NLP) has seen tremendous growth and development, becoming an integral part of various applications, from chatbots to sentiment analysis. One of the foundational steps in NLP is text pre-processing, which involves cleaning and preparing raw text data for further analysis or model training. Proper text pre-processing can significantly impact the performance and accuracy of NLP models.

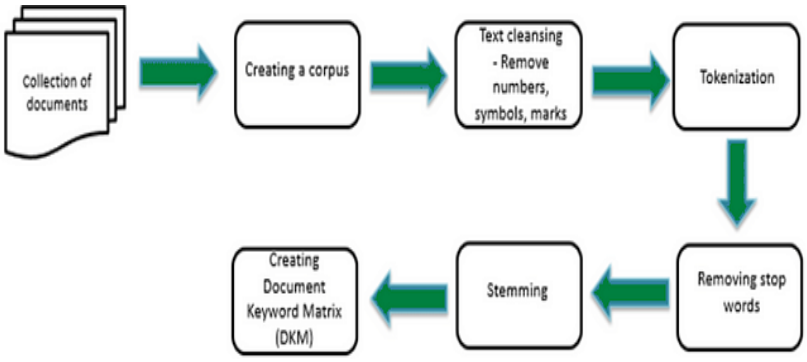
Why Text Pre-processing is Important?

Raw text data is often noisy and unstructured, containing various inconsistencies such as typos, slang, abbreviations, and irrelevant information.

- Pre-processing helps in: Improving Data Quality: Removing noise and irrelevant information ensures that the data fed into the model is clean and consistent.
- Enhancing Model Performance: Well-pre-processed text can lead to better feature extraction, improving the performance of NLP models.
- Reducing Complexity: Simplifying the text data can reduce the computational complexity and make the models more efficient.

Text Pre-processing Techniques

Text pre-processing refers to a series of techniques used to clean, transform and prepare raw textual data into a format that is suitable for NLP or ML tasks. The goal of text pre-processing is to enhance the quality and usability of the text data for subsequent analysis or modelling.



Text Pre-processing Technique in NLP

1. Text Cleaning

Remove Noise: Eliminate characters, symbols, or irrelevant data such as special characters, HTML tags, or URLs that don't contribute to the analysis.
Example: Remove HTML tags or URLs using regex or libraries like BeautifulSoup.

- Remove Punctuation: Punctuation marks such as periods, commas, or exclamation marks are often removed unless they have specific significance (e.g., sentiment analysis).
- Remove Non-Alphanumeric Characters: Remove numbers or any non-textual characters if they are irrelevant to the task at hand.

import re text = re.sub(r'[^\w\s]', '', text) # Removes punctuation

2. Tokenization

Tokenization involves breaking the text into smaller units, such as words or sentences.

- Word Tokenization: Splitting text into words is often done using libraries like NLTK, spaCy, or transformers.
- Sentence Tokenization: Breaking text into sentences (e.g., for sentiment analysis on each sentence).

from nltk.tokenize import word_tokenize words = word_tokenize(text)

3. Lowercasing

Convert all text to lowercase to standardize words, as "The" and "the" should be treated the same in NLP tasks.

text = text.lower()

4. Removing Stop Words

Stop words are common words (like "the," "is," "and," etc.) that don't add significant meaning to the text and can be removed. Libraries like NLTK provide a list of stop words for multiple languages.

from nltk.corpus import stopwords stop_words = set(stopwords.words('english')) words = [word for word in words if word not in stop_words]

5. Stemming

Stemming reduces words to their root form (e.g., "running" → "run"). Stemming algorithms like PorterStemmer or SnowballStemmer are used to achieve this.

from nltk.stem import PorterStemmer stemmer = PorterStemmer() words = [stemmer.stem(word) for word in words]

6. Lemmatization

Lemmatization is a more sophisticated form of reducing words to their base form (e.g., "better" → "good") based on context and part-of-speech tagging. Libraries like spaCy or WordNetLemmatizer in NLTK are used for lemmatization.

from nltk.stem import WordNetLemmatizer lemmatizer = WordNetLemmatizer() words = [lemmatizer.lemmatize(word) for word in words]

7. Removing Rare Words and Out-of-Vocabulary Terms

Removing extremely rare words or terms that occur too infrequently can help reduce noise. You may set a frequency threshold and discard words with a frequency lower than that threshold.

from collections import Counter word_freq = Counter(words) words = [word for word in words if word_freq[word] > threshold]

8. Handling Word Embeddings or Vectorization

Convert text into numerical representations that machine learning models can process:

- Bag of Words (BoW): Represents text by counting the frequency of words.
- TF-IDF (Term Frequency-Inverse Document Frequency): Weighs terms based on their frequency in a document relative to their frequency across all documents.
- Word2Vec / GloVe / FastText: Pre-trained word embeddings represent words as vectors based on their semantic meaning.

from sklearn.feature_extraction.text import TfidfVectorizer vectorizer = TfidfVectorizer() X = vectorizer.fit_transform(documents)

9. Handling Imbalanced Data (Optional)

If your data has an imbalanced distribution of classes (e.g., positive vs. negative sentiment), you may need to apply techniques like:

- Resampling: Oversample the minority class or undersample the majority class.
- Class Weights: Assign higher weights to the minority class in classification algorithms.

10. Part-of-Speech (POS) Tagging (Optional)

POS tagging can provide more context to words by classifying them into parts of speech, such as noun, verb, adjective, etc. This is important in tasks like named entity recognition (NER) or information extraction.

import spacy nlp = spacy.load('en_core_web_sm') doc = nlp(text) for token in doc: print(token.text, token.pos_)

11. Named Entity Recognition (NER) (Optional)

Identifying entities (e.g., names of people, organizations, locations) from text can be important in tasks like information extraction.

entities = [(ent.text, ent.label_) for ent in doc.ents]

12. Handling Negation (Optional)

Negation handling is important in tasks like sentiment analysis, where negating words (e.g., "not happy") changes the meaning of the sentence.
Some advanced models account for negation using custom techniques or embeddings.

13. Dealing with Spelling Errors (Optional)

Text may contain misspelled words, and correcting them can improve model performance. Libraries like TextBlob or pyspellchecker can be used for spell-checking.
from spellchecker import SpellChecker spell = SpellChecker() corrected_words = [spell.correction(word) for word in words]

What is stemming?

Stemming is a text pre-processing technique used in natural language processing (NLP) to reduce words to their root or base form. The goal of stemming is to simplify and standardize words, which helps improve the performance of information retrieval, text classification, and other NLP tasks. By transforming words to their stems, NLP models can treat different forms of the same word as a single entity, reducing the complexity of the text data.

For example, stemming would reduce the words “running,” “runner,” and “runs” to their stem “run.” This allows the NLP model to recognize that these words share a common concept or meaning, even though their forms are different.

Stemming algorithms typically work by removing or replacing word suffixes or prefixes, based on a set of predefined rules or heuristics. Some common stemming algorithms include the Porter Stemmer, Lancaster Stemmer, and Snowball Stemmer.

Reasons for using stemming:

- Text simplification: Stemming helps simplify text data by reducing words to their base forms, making it easier for NLP models to process and analyse the text.
- Improved model performance: By reducing word variations, stemming can lead to better model performance in tasks such as text classification, sentiment analysis, and information retrieval.
- Standardization: Stemming standardizes words, which helps in comparing and matching text data across different sources and contexts.

Principles of Stemming Algorithms:

Stemming algorithms operate based on linguistic rules and heuristics to strip affixes from words and obtain their stems. The goal is to map related words to the same root, thereby simplifying text processing and enhancing computational efficiency.

Application of Stemming

In information retrieval, text mining SEOs, Web search results, indexing, tagging systems, and word analysis, stemming is employed. For instance, a Google search for prediction and predicted returns comparable results.

Common Stemming Techniques:

- **Porter Stemmer:** Developed by Martin Porter in the 1980s, the Porter Stemmer is one of the oldest and most widely used stemming algorithms. It is designed primarily for English words and applies a series of rules to remove suffixes and transform words to their base form.
- **Snowball Stemmer:** Also known as the Porter2 Stemmer, the Snowball Stemmer is an extension of the Porter algorithm with support for multiple languages. It employs a more systematic approach and can handle stemming tasks in languages beyond English, including French, German, and Spanish.
- **Lancaster Stemmer:** Created by Chris Paice at Lancaster University, the Lancaster Stemmer is known for its aggressive stemming strategy. It applies a set of heuristic rules to truncate words aggressively, often resulting in shorter stems compared to other algorithms.

Porter stemmer

Compared to the Lovins stemmer, the Porter stemming algorithm uses a more mathematical stemming algorithm. Essentially, this stemmer classifies every character in a given token as either a consonant (c) or vowel (v), grouping subsequent consonants as C and subsequent vowels as V. The stemmer thus represents every word token as a combination of consonant and vowel groups. Once enumerated this way, the stemmer runs each word token through a list of rules that specify ending characters to remove according to the number of vowel-consonant groups in a token. Because English itself follows general but not absolute lexical rules, the Porter stemmer algorithm's systematic criterion for determining suffix removal can return errors.

Python NLTK contains a built-in Porter stemmer function. This code deploys the Porter stemming algorithm on the tokenized Shakespeare quotation:

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
porter_stemmer = PorterStemmer()
text = "Love looks not with the eyes but with the mind, and therefore is winged Cupid painted blind."
words = word_tokenize(text)
stemmed_words = [porter_stemmer.stem(word) for word in words]
```

OUTPUT:

Stemmed: ['love', 'look', 'not', 'with', 'the', 'eye', 'but', 'with', 'the', 'mind', ',', 'and', 'therefor', 'is', 'wing', 'cupid', 'paint', 'blind', '.']

As with Lovins, Porter correctly changes verb conjugations and noun pluralization's. While lacking Lovins' other malformed stems (for example, love to lov), the Porter stemming algorithm nevertheless erroneously removes -e from the end of therefore.

Per the Porter stemmer's consonant-vowel grouping method, therefore is represented as CVCVCVCV, or C(VC)3V, with the exponent signifying repetitions of consonant-vowel groups.

One of the algorithm's final steps states that, if a word has not undergone any stemming and has an exponent value greater than 1, -e is removed from the word's

ending (if present). Therefore, exponent value equals 3, and it contains none of the suffixes listed in the algorithm's other conditions. Thus, therefore becomes therefor.

Admittedly, this is the Porter stemmer's only error, perhaps testifying to why it is the most widely adopted stemming algorithm. Indeed, the Porter stemmer has served as a foundation for subsequent stemming algorithms.

Snowball stemmer

Snowball stemmer is an updated version of the Porter stemmer. While it aims to enforce a more robust set of rules for determining suffix removal, it nevertheless remains prone to many of the same errors. Much like the Porter stemmer, Python NLTK contains a built-in Snowball stemmer function:

```
from nltk.stem.snowball import SnowballStemmer
from nltk.tokenize import word_tokenize
stemmer = SnowballStemmer("english", True)
text = "There is nothing either good or bad but thinking makes it so."
words = word_tokenize(text)
stemmed_words = [stemmer.stem(word) for word in words]
```

The produces the same output of the Shakespeare text as the Porter stemmer, incorrectly reducing therefore to therefor:

OUTPUT:

Stemmed: ['love', 'look', 'not', 'with', 'the', 'eye', 'but', 'with', 'the', 'mind', ',', 'and', 'therefor', 'is', 'wing', 'cupid', 'paint', 'blind', '.']

The Snowball stemmer differs from Porter in two main ways. First, while the Lovins and Porter stemmers only stem English words, the Snowball stemmer can stem texts in a number of other Roman script languages, such as Dutch, German, French, and even Russian. Second, the Snowball stemmer, when implemented via Python NLTK library, can ignore stop-words. Stop-words are a non-universal collection of words that are removed from a dataset during pre-processing. The Snowball stemmer's predefined stop-list contains words without a direct conceptual definition and that serve more a grammatical than semantic function. Stop-words included in the Snowball stemmer English stop-list include the, a, being, and the like.

Lancaster stemmer

Many sources describe the Lancaster stemmer also known as the Paice stemmer as the most aggressive of English language stemmers. The Lancaster stemmer contains a list of over 100 rules that dictate which ending character strings, if present, to replace with other strings, if any. The stemmer iterates through each word token, checking it against all the rules. If the token's ending string matches that of a rule, the algorithm enacts the rule's described operation and then runs the new, transformed word through all of the rules again. The stemmer iterates through all of the rules until a given token passes them all without being transformed.

Though unavailable in Python NLTK, the Lancaster stemmer is available in stemming library:

```
from stemming.paicehusk import stem
from nltk.tokenize import word_tokenize
```

```
text = "Love looks not with the eyes but with the mind, and therefore is winged Cupid painted blind."
words = word_tokenize(text)
stemmed_words = [stem(word) for word in words]
```

OUTPUT:

Stemmed: ['Lov', 'look', 'not', 'with', 'the', 'ey', 'but', 'with', 'the', 'mind', ',', 'and', 'theref', '', 'wing', 'Cupid', 'paint', 'blind', '.']

Clearly, the Lancaster stemmer's iterative approach is the most aggressive of the stemmers, as shown with there. First, the Lancaster stemmer has the rule "e1>". This rule removes the single-character- e with no replacement. After the algorithm strips -e from therefore, it runs the new therefor through each rule. The newly transformed word fits the rule "ro2>." This rule removes the two-character suffix -or with no replacement. The resulting stem their fits none of the algorithms other rules and so is returned as the stemmed base. Unlike Lovins, the Lancaster algorithm has no means of accounting for malformed words.

Advantages

- Improves search accuracy: Stemming links related words, helping to identify information of interest.
- Reduced dimensionality: By collapsing multiple forms of a word into a single representation, stemming reduces dimensionality and can make statistical processing easier.
- Quick processing: Stemming algorithms are generally straightforward and fast, which speeds up the processing time for large volumes of text.

Disadvantages

- Over-stemming: Sometimes, stemming can be too aggressive, resulting in different words being reduced to the same stem even though they have different meanings (e.g., "metaphor" and "metaphysical" might stem to "meta").
- Under-stemming: At other times, stemming might not be aggressive enough, failing to conflate words that are practically the same (e.g., "ready" and "readiness" might not show as related).
- Difficulty with irregular conjugation: If the word is in a form not included in the pre-defined set of suffixes, the algorithm may not recognize it or may stem it improperly.

Lemmatization

Lemmatization is a fundamental **text pre-processing technique** widely applied in **Natural Language Processing (NLP)** and machine learning. Serving a purpose akin to stemming, lemmatization seeks to distill words to their foundational forms. In this linguistic refinement, the resultant base word is referred to as a "lemma." The article aims to explore the use of lemmatization and demonstrates how to perform

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So, it links words with similar meanings to one word. Text pre-processing includes both Stemming as well as lemmatization. Many times, people find these two terms confusing. Some treat these two as the same. Lemmatization

is preferred over Stemming because lemmatization does morphological analysis of the words.

Lemmatization Techniques

Lemmatization techniques in natural language processing (NLP) involve methods to identify and transform words into their base or root forms, known as lemmas. These approaches contribute to text normalization, facilitating more accurate language analysis and processing in various NLP applications. Three types of lemmatization techniques are:

1. Rule Based Lemmatization

Rule-based lemmatization involves the application of predefined rules to derive the base or root form of a word. Unlike machine learning-based approaches, which learn from data, rule-based lemmatization relies on linguistic rules and patterns.

Here's a simplified example of rule-based lemmatization for English verbs:

Rule: For regular verbs ending in "-ed," remove the "-ed" suffix.

Example:

- Word: "walked"
- Rule Application: Remove "-ed"
- Result: "walk

This approach extends to other verb conjugations, providing a systematic way to obtain lemmas for regular verbs. While rule-based lemmatization may not cover all linguistic nuances, it serves as a transparent and interpretable method for deriving base forms in many cases.

2. Dictionary-Based Lemmatization

Dictionary-based lemmatization relies on predefined dictionaries or lookup tables to map words to their corresponding base forms or lemmas. Each word is matched against the dictionary entries to find its lemma. This method is effective for languages with well-defined rules.

Suppose we have a dictionary with lemmatized forms for some words:

- 'running' -> 'run'
- 'better' -> 'good'
- 'went' -> 'go'

When we apply dictionary-based lemmatization to a text like "I was running to become a better athlete, and then I went home," the resulting lemmatized form would be: "I was run to become a good athlete, and then I go home."3. Machine Learning-Based Lemmatization

Machine learning-based lemmatization leverages computational models to automatically learn the relationships between words and their base forms. Unlike rule-based or dictionary-based approaches, machine learning models, such as neural networks or statistical models, are trained on large text datasets to generalize patterns in language.

Example:

Consider a machine learning-based lemmatize trained on diverse texts. When encountering the word 'went,' the model, having learned patterns, predicts the base form as 'go.' Similarly, for 'happier,' the model deduces 'happy' as the lemma. The advantage lies in the model's ability to adapt to varied linguistic nuances and handle irregularities, making it robust for lemmatizing diverse vocabularies.

Advantages

- **Increased Accuracy:** By supplying the true root word, lemmatization improves accuracy over stemming and facilitates more sophisticated language comprehension and analysis.
- **Context Preservation:** By returning words to their root forms in accordance with their intended meaning within a sentence, it preserves the context and meaning of those words.
- **Enhanced Text Normalization:** Lemmatization reduces variances and improves text analysis and information retrieval precision by normalizing words to their dictionary form.
- **Improved Search Engine Results:** It increases search relevancy by clustering words with similar meanings together to produce better search results.

Disadvantages

- **Computational Complexity:** Compared to stemming, lemmatization requires access to dictionary resources and part-of-speech tagging, which can lead to longer processing times and higher resource consumption.
- **Loss of Speed:** Lemmatization takes longer than stemming, which could be a drawback in situations where processing in real-time is essential.
- **Dependency on Language Resources:** Lemmatization is highly dependent on lexical databases and dictionaries, which may not be able to cover all complexities or specialized terminology in the language.
- **Over-Lemmatization:** There's a chance that over-normalization could cause some words to become too generalized and lose their unique meaning, which could affect text analysis or sentiment classification accuracy.

Diving Chunks

As language models become more sophisticated and capable of handling increasingly complex tasks, a major challenge arises: how do we efficiently structure vast amounts of unstructured text data for these models to process? The answer lies in **chunking -** a technique used to break down large documents into smaller, meaningful segments that preserve semantic meaning while optimizing text for model performance.

But chunking is not just about breaking down text; it's about implementing thoughtful strategies to create coherent pieces of information that are easier for NLP models to analyze. In complex scenarios like information retrieval, question answering, and text summarization, maintaining context across chunks becomes crucial. This is where **overlapping chunks** come into play, allowing us to retain critical context and build better-performing NLP systems.

In this guide, we'll take an in-depth look at chunking strategies, the role of overlapping, and how to implement these techniques using the powerful open-source library, **LangChain**. LangChain simplifies the entire process by providing tools for text chunking, embedding, and document retrieval, making it ideal for building advanced NLP systems. By the end of this article, you'll understand how to optimize text data for any downstream NLP application.

What is Chunking?

Chunking is the process of breaking down a large text document into smaller, coherent units called "chunks." Each chunk should represent a complete thought or

segment of text, preserving the semantic meaning of the content. This makes it easier for machine learning models to process text, particularly when dealing with long documents or multi-page reports.

Why is Chunking Important in NLP?

When working with large datasets, the need for chunking arises from various considerations, including:

1. **Memory Management**: Large texts can overwhelm memory, especially in low-resource environments. Chunking helps break down data into manageable pieces.
2. **Improved Model Efficiency**: Smaller chunks reduce computational complexity, making it easier for models to process large volumes of data quickly.
3. **Context Management**: Proper chunking helps retain the context within each segment, which is essential for understanding relationships between parts of the text.
4. **Better Retrieval and Search**: Chunks make it easier to search for and retrieve specific pieces of information, boosting the accuracy of information retrieval tasks.
5. **Parallel Processing**: Smaller chunks can be processed independently, enabling parallelization, which speeds up model training and inference.

Understanding Different Chunking Strategies

Different chunking strategies are applied depending on the text type and the requirements of the task:

1. **Fixed-size Chunking**: Splitting text into equal-sized chunks based on a predetermined number of words, characters, or tokens. This method works well for homogeneous text without semantic shifts.
2. **Semantic Chunking**: Breaking text based on semantic boundaries, such as paragraphs, sections, or topic shifts, ensuring that each chunk captures a complete thought or idea.
3. **Sliding Window Chunking**: A window of a defined size slides over the text, creating overlapping chunks to retain context between adjacent segments.
4. **Sentence-based Chunking**: Splits text at sentence boundaries, ensuring that no sentence is broken across chunks, which is particularly useful for question answering and text generation tasks.

Why Overlapping is Necessary in Chunking

One of the primary challenges with chunking is the potential loss of critical context at the boundaries. For example, consider a long document discussing various aspects of a complex topic. If chunks are created without overlap, the model might lose key contextual information between adjacent segments, reducing its ability to understand the complete context. This is where **overlapping** becomes useful.

What is Overlapping?

Overlapping refers to creating chunks that share a portion of text with adjacent chunks. By doing so, overlapping ensures that no vital information is lost at the boundaries, making it particularly useful in scenarios like:

- **Question Answering**: For accurately answering questions, the model needs to consider adjacent chunks to capture all relevant information.
- **Summarization**: When generating summaries, overlapping chunks help maintain coherence by linking related ideas together.

- **Information Retrieval**: In retrieval-augmented generation (RAG) systems, overlapping improves the accuracy and relevance of results by ensuring no information is missed.

Types of Overlapping Strategies

1. **Fixed-size Overlapping Chunks**: Each chunk has a fixed size, and a defined number of words or sentences overlap between chunks.
2. **Variable-size Overlapping Chunks**: Chunks vary in size based on semantic or syntactic cues but still maintain overlap to preserve context.
3. **Sliding Window Overlapping**: A sliding window approach creates overlapping chunks by moving a few tokens, words, or sentences at each step.

Types of Chunking

- Chunking Up: - **We don't go into great detail here; instead, we're content with a high-level overview. It only serves to provide us with a quick overview of the facts.**
- Chunking Down: - **Unlike the previous method of Chunking, chunking down allows us to obtain more detailed data.**

Consider "chunking up" if you only need an insight; otherwise, "chunking down" is preferable.

Regular Expressions

To learn how to implement Chunking, some knowledge about Regular Expressions (regex) is required. Regex is a kind of instruction in which we define what are the types of substring that needs to be selected from a text. There is some specific

format rules which are defined for it. Let's learn an introduction to them. Here's where you can learn more about Regular Expressions:

Symbol	Meaning	Example
*	The preceding character can occur zero or more times meaning that the preceding character may or may not be there.	ab* matches all inputs starting with ab and then followed by zero or more numbers of b's. The patter will match ab, abb, abbb, and so on.
+	The preceding character should occur at least once	a+ matches a, aa, aaa, and so on.
?	The preceding character may not occur at all or occur only once meaning the preceding character is optional	ab? matches ab, abb, but not abbb, and so on.

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence(can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrence	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"
()	Capture and group	

Chunking in Python

The high-level idea is that first; we tokenize our text. Now there is a utility in NLTK which tags the words; pos_tag, which attaches a tag to the words, for example, Verb conjunction etc.

Then with the help of these tags, we can perform Chunking. If we want to select verbs, we can write a grammar that selects the words with a grammar tag.

Let's understand the code:

text = word_tokenize("And now for something completely different")
nltk.pos_tag(text)

Output: [('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'), ('completely', 'RB'), ('different', 'JJ')]

'CC' is connecting conjunction and so on. Here you can understand the meaning of some tags that are there in NLTK:

POS	Meaning
VB	Verb in its base form
VBD	Verb in its past tense
VBG	Verb in its present tense
VBN	Verb in its part participle form
VBP	Verb in its present tense but not in third person singular
VBZ	Verb in its present tense and is third person singular

After this tagging, we can define our rule and perform verb chunking or noun Chunking etc.

Text Classification

Text data is one of the most common types of data that companies use today, but because it doesn't have a clear structure, it can be difficult and time-consuming to extract insights from text data. Dealing with text data comes under **Natural Language Processing**, one of the subfields of **artificial intelligence**.

Natural Language Processing (NLP) is a field of computer science and artificial intelligence that looks at how computers interact with human languages and how to program computers to process and analyse large amounts of natural language data.

NLP is used in many different ways, such as to answer questions automatically, generate summaries of texts, translate texts from one language to another, etc. NLP research is

also conducted in areas such as cognitive science, linguistics, and psychology. Text classification is one such use case for NLP.

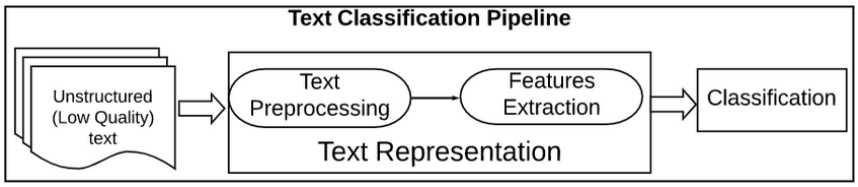
This blog will explore **text classification use cases**. It also contains an end-to-end example of how to build a text pre-processing pipeline followed by a text classification model in Python.

What is Text Classification?

Text classification is a common NLP task used to solve business problems in various fields. The goal of text classification is to categorize or predict a class of unseen text documents, often with the help of supervised machine learning.

Similar to a classification algorithm that has been trained on a tabular dataset to predict a class, text classification also uses **supervised machine learning**. The fact that text is involved in text classification is the main distinction between the two.

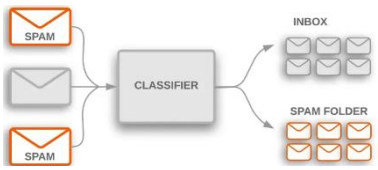
You can also perform text classification without using supervised machine learning. Instead of algorithms, a manual rule-based system can be designed to perform the task of text classification. We'll compare and review the pros and cons of rule-based and machine-learning-based text classification systems in the next section.



Text Classification Use-Cases and Applications

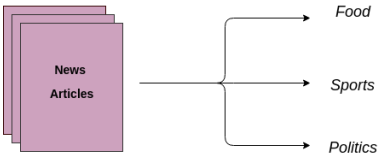
Spam classification

There are many practical use cases for text classification across many industries. For example, a spam filter is a common application that uses text classification to sort emails into spam and non-spam categories.



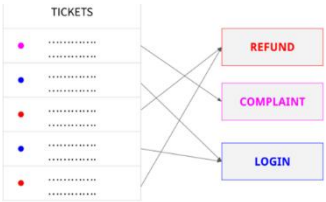
Classifying news articles and blogs

Another use case is to automatically assign text documents into predetermined categories. A supervised machine learning model is trained on labelled data, which includes both the raw text and the target. Once a model is trained, it is then used in production to obtain a category (label) on the new and unseen data (articles/blogs written in the future).

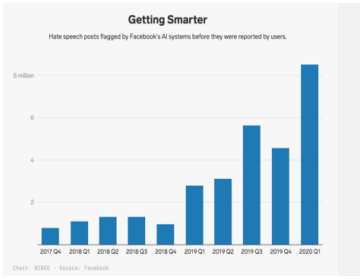


Categorize customer support requests

A company might use text classification to automatically categorize customer support requests by topic or to prioritize and route requests to the appropriate department.



Hate speech detection



With over 1.7 billion daily active users, Facebook inevitably has content created on the site that is against the rules. Hate speech is included in this undesirable content.

Facebook tackles this issue by requesting a manual review of postings that an **AI text classifier** has identified as hate speech. Postings that were flagged by AI are examined in the same manner as posts that users have reported. In fact, in just the first three months of 2020, the platform removed 9.6 million items of content that had been classified as hate speech.

Types of Text Classification Systems

There are mainly two types of text classification systems; rule-based and machine learning-based text classification.

Rule-based text classification

Rule-based techniques use a set of manually constructed language rules to categorize text into categories or groups. These rules tell the system to classify text into a particular category based on the content of a text by using semantically relevant textual elements. An antecedent or pattern and a projected category make up each rule.
For example, imagine you have tons of new articles, and your goal is to assign them to relevant categories such as Sports, Politics, Economy, etc.

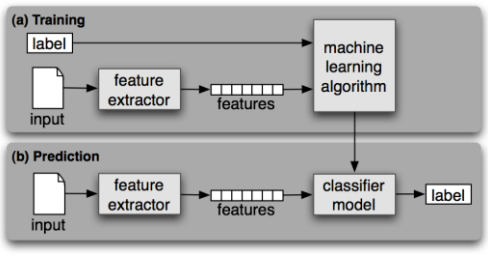
With a rule-based classification system, you will do a human review of a couple of documents to come up with linguistic rules like this one:

If the document contains words such as *money*, *dollar*, *GDP*, or *inflation*, it belongs to the Politics group (class).

Rule-based systems can be refined over time and are understandable to humans. However, there are certain drawbacks to this strategy.
These systems, to begin with, demand in-depth expertise in the field. They take a lot of time since creating rules for a complicated system can be difficult and frequently necessitates extensive study and testing.

Given that adding new rules can alter the outcomes of the pre-existing rules, rule-based systems are also challenging to maintain and do not scale effectively.

Machine learning-based text classification



Machine learning-based text classification is a supervised machine learning problem. It learns the mapping of input data (raw text) with the labels (also known as target variables). This is similar to non-text classification problems where we train a supervised classification algorithm on a tabular dataset to predict a class, with the exception that in text classification, the input data is raw text instead of numeric features.

Like any other supervised machine learning, text classification machine learning has two phases; training and prediction.

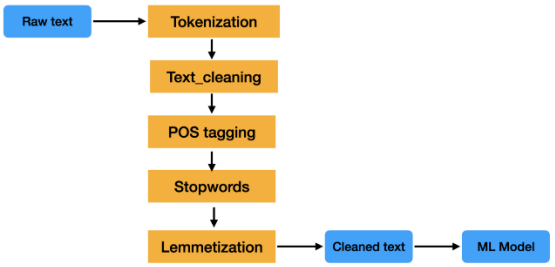
Training phase

A supervised machine learning algorithm is trained on the input-labelled dataset during the training phase. At the end of this process, we get a trained model that we can use to obtain predictions (labels) on new and unseen data.

Prediction phase

Once a machine learning model is trained, it can be used to predict labels on new and unseen data. This is usually done by deploying the best model from an earlier phase as an API on the server.

Text Pre-processing Pipeline



Pre-processing text data is an important step in any natural language processing task. It helps in cleaning and preparing the text data for further processing or analysis.
A text pre-processing pipeline is a series of processing steps that are applied to raw text data in order to prepare it for use in natural language processing tasks.

The steps in a text pre-processing pipeline can vary, but they typically include tasks such as tokenization, stop word removal, stemming, and lemmatization. These steps help reduce the size of the text data and also improve the accuracy of NLP tasks such as text classification and information extraction.

Text data is difficult to process because it is unstructured and often contains a lot of noise. This noise can be in the form of misspellings, grammatical errors, and non-standard formatting. A text pre-processing pipeline aims to clean up this noise so that the text data can be more easily analysed.

Feature Extraction

The two most common methods for extracting feature from text or in other words converting text data (strings) into numeric features so machine learning model can be trained are: Bag of Words (a.k.a CountVectorizer) and Tf-IDF.

Bag of Words

A **Bag of Words** (BoW) model is a simple way of representing text data as numeric features. It involves creating a vocabulary of known words in the corpus and then creating a vector for each document that contains counts of how often each word appears.

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

TF-IDF

TF-IDF stands for term frequency-inverse document frequency, and it is another way of representing text as numeric features. There are some shortcomings of the **Bag of Words (BoW)** model that Tf-IDF overcomes.

The TF-IDF model is different from the bag of words model in that it takes into account the frequency of the words in the document, as well as the inverse document frequency. This means that the TF-IDF model is more likely to identify the important words in a document than the bag of words model.

Unit -5 Computer Vision With OpenCV

Object Detection

Object detection in the domains of computer vision is the task of detecting and locating multiple objects within an image or a video frame. While object recognition tells us what objects are in the image but does not always give us their location, object detection provides a very accurate location of each object by drawing parameterized rectangles, and the bounding boxes. This task is useful for several purposes such as security, monitoring, self-driving cars, image search, etc.

Object detection primarily aims to answer two critical questions about any image: "Which objects are present?" and "Where are these objects situated?" This process involves both object classification and localization:

- **Classification:** This step determines the category or type of one or more objects within the image, such as a dog, car, or tree.
- **Localization:** This involves accurately identifying and marking the position of an object in the image, typically using a bounding box to outline its location.

Key Components of Object Detection

1. Image Classification

Image classification assigns a label to an entire image based on its content. While it's a crucial step in understanding visual data, it doesn't provide information about the object's location within the image.

2. Object Localization

Object localization goes a step further by not only identifying the object but also determining its position within the image. This involves drawing bounding boxes around the objects.

3. Object Detection

Object detection merges image classification and localization. It detects multiple objects in an image, assigns labels to them, and provides their locations through bounding boxes.

Key features of object detection:

- **Localization and Classification:** At the same time as estimating the rectangular spatial coverage and categorizing each detected object.
- **Multiple Object Detection:** Allowing the algorithm to identify the presence of one or several objects in the same picture or frame.
- **Precision and Recall:** Measures commonly employed in assessing the performance of an algorithm for a detection model about how well it locates objects in a scene.
- **Speed and Efficiency:** Advanced computation techniques to support real-time interaction for screen displays, high-resolution pictures and video stream outputs.
- **Object Tracking:** Including time information in the tracking process to identify objects in consecutive frames, thus making the tracking stable in dynamic environments.

How Object Detection works?

The general working of object detection is:

1. **Input Image:** the object detection process begins with image or video analysis.
2. **Pre-processing:** image is pre-processed to ensure suitable format for the model being used.
3. **Feature Extraction:** CNN model is used as feature extractor, the model is responsible for dissecting the image into regions and pulling out features from each region to detect patterns of different objects.
4. **Classification:** Each image region is classified into categories based on the extracted features. The classification task is performed using SVM or other neural network that computes the probability of each category present in the region.
5. **Localization:** Simultaneously with the classification process, the model determines the bounding boxes for each detected object. This involves calculating the coordinates for a box that encloses each object, thereby accurately locating it within the image.
6. **Non-max Suppression:** When the model identifies several bounding boxes for the same object, non-max suppression is used to handle these overlaps. This technique keeps only the bounding box with the highest confidence score and removes any other overlapping boxes.
7. **Output:** The process ends with the original image being marked with bounding boxes and labels that illustrate the detected objects and their corresponding categories.

Techniques in Object Detection

Traditionally, the task of object detection relied on manual feature extraction and classification. Some of the tradition methods are:

1. Haar Cascades
2. Histogram of Oriented Gradients (HOG)
3. SIFT (Scale-Invariant Feature Transform)

Introduction to OpenCV

OpenCV is one of the most popular computer vision libraries. If you want to start your journey in the field of computer vision, then a thorough understanding of the concepts of OpenCV is of paramount importance.

To understand the basic functionalities of Python OpenCV module, we will cover the most basic and important concepts of OpenCV intuitively:

1. Reading an image
2. Extracting the RGB values of a pixel
3. Extracting the Region of Interest (ROI)
4. Resizing the Image
5. Rotating the Image
6. Drawing a Rectangle
7. Displaying text

This is the original image that we will manipulate throughout the explanation and code for the implementation, we need to install the OpenCV library using the following command:

pip install opencv-python

Reading an Image

First of all, we will import cv2 module and then read the input image using cv2's imread() method. Then extract the height and width of the image.

```
# Importing the OpenCV library
import cv2
# Reading the image using imread() function
image = cv2.imread('image.jpg')
# Extracting the height and width of an image
h, w = image.shape[:2]
# Displaying the height and width
print("Height = {}, Width = {}".format(h, w))
```

Output: **Height = 1603, Width = 2400**

Extracting the RGB Values of a Pixel

Now we will focus on extracting the RGB values of an individual pixel. OpenCV arranges the channels in BGR order. So the 0th value will correspond to the Blue pixel and not the Red.

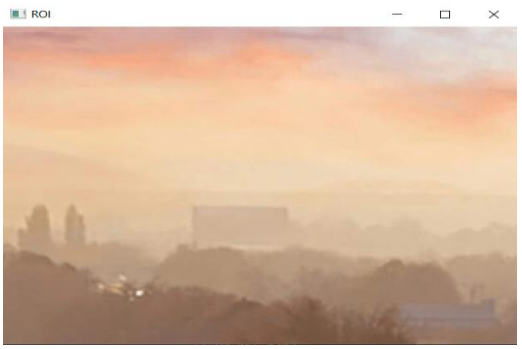
```
# Extracting RGB values.
# Here we have randomly chosen a pixel
# by passing in 100, 100 for height and width.
(B, G, R) = image[100, 100]
# Displaying the pixel values
print("R = {}, G = {}, B = {}".format(R, G, B))
# We can also pass the channel to extract
# the value for a specific channel
B = image[100, 100, 0]
print("B = {}".format(B))
```

Output :- R = 211, G = 172, B = 165B = 165

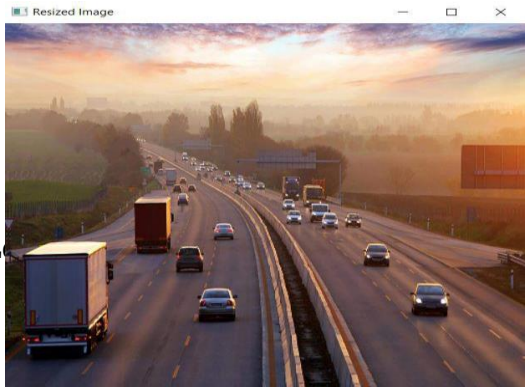
Extracting the Region of Interest (ROI)

Sometimes we want to extract a particular part or region of an image. This can be done by slicing the pixels of the image.

```
# We will calculate the region of interest
# by slicing the pixels of the image
roi = image[100 : 500, 200 : 700]
cv2.imshow("ROI", roi)
cv2.waitKey(0)
```



Resizing the Image



We can also resize an image in Python using resize() function of the cv2 module and pass the input image and resize pixel value.

```
# resize() function takes 2 parameters,
# the image and the dimensions
resize = cv2.resize(image, (500, 500))
cv2.imshow("Resized Image", resize)
cv2.waitKey(0)
```

The problem with this approach is that the aspect ratio of the image is not maintained. So we need to do some extra work in order to maintain a proper aspect ratio.

```
# Calculating the ratio
ratio = 800 / w
# Creating a tuple containing width and height
dim = (800, int(h * ratio))
# Resizing the image
resize_aspect = cv2.resize(image, dim)
cv2.imshow("Resized Image", resize_aspect)
cv2.waitKey(0)
```

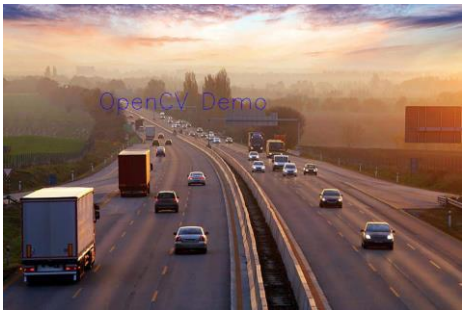
Drawing a Rectangle

We can draw a rectangle on the image using rectangle() method. It takes in 5 arguments:

- Image
- Top-left corner co-ordinates
- Bottom-right corner co-ordinates
- Color (in BGR format)
- Line width

```
# We are copying the original image,
# as it is an in-place operation.
output = image.copy()
# Using the rectangle() function to create a rectangle.
rectangle = cv2.rectangle(output, (1500, 900),(600, 400), (255, 0, 0), 2)
```

Displaying text



It is also an in-place operation that can be done using the putText() method of OpenCV module. It takes in 7 arguments:

- Image
- Text to be displayed
- Bottom-left corner co-ordinates, from where the text should start
- Font
- Font size
- Color (BGR format)
- Line width

```
# Copying the original image
output = image.copy()
# Adding the text using putText() function
text = cv2.putText(output, 'OpenCV Demo', (500, 550), cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 0, 0), 2)
```

Haar Cascades for Object Detection

OpenCV is the huge open-source library for computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human.

Object Detection

Object Detection is a computer technology related to computer vision, image processing, and deep learning that deals with detecting instances of objects in images and videos. We will do object detection in this article using something known as **haar cascades.**

Haar Cascades

Paul Viola and Michael Jones proposed Haar Cascade Algorithm, which is productively used for Object Detection. This Algorithm is based on a Machine Learning approach in which lots of images are used, whether positive or negative, to train the classifier.

- **Positive Images:** Positive Images are a type of image that we want our classifier to identify.
- **Negative Images:** Negative Images are a type of image that contains something else, i.e., it does not contain the objects we want to detect.

The first real-time face detector also used the Haar classifiers, which we are introducing here. Finding objects in pictures and videos is done by a machine learning programme known as a Haar classifier or a Haar cascade classifier.

Steps to download the requirements below

Run The following command in the terminal to install opencv.

- pip install opencv-python

Run the following command to in the terminal install the Matplotlib.

- pip install matplotlib

Haar features and Haar cascades were the tools you must not ignore for object detection. Even today, they are very useful object detectors because they are lightweight. In this post, you will learn about the Haar cascade and how it can detect objects.

What are Haar Features and Haar Cascade?

Since the technique developed by **Paul Viola and Michael Jones in 2001**, Haar features and Haar cascades have revolutionized object detection. They have become integral components in various applications, ranging from facial recognition to real-time object detection.

Haar features are extracted from rectangular areas in an image. The feature's value is based on the pixel intensities. Usually, it is calculated using a sliding window, and the area within the window is partitioned into two or more rectangular areas. Haar feature is the difference in the sum of pixel intensities between these areas.

It is believed that an object's presence will distort the variation of pixel intensity. For example, the background is usually in a uniform pattern, in which a foreground object will not fit. By checking the pixel intensity between neighbouring rectangular areas, you should be able to notice a difference. Hence it is indicative of the object's presence.

For the efficiency of calculation, the rectangular areas in Haar features are usually parallel to the edges of the image rather than tilted. However, we can use multiple sizes and shapes of rectangles to capture different features and scale variations of an object. Therefore, the key strength of Haar features lies in their ability to represent three patterns:

- **Edges:** Either vertical or horizontal due to how we oriented the rectangular area. They are useful for identifying boundaries between different image regions.
- **Lines:** The diagonal edges in an image. They are useful for identifying lines and contours in objects.
- **Center-surrounded features**: This detects the changes in intensity between the center of a rectangular region and its surrounding area. This is useful to identify objects with a distinct shape or pattern.

Haar cascade combines multiple Haar features in a hierarchy to build a classifier. Instead of analysing the entire image with each Haar feature, cascades break down the detection process into stages, each consisting of a set of features.

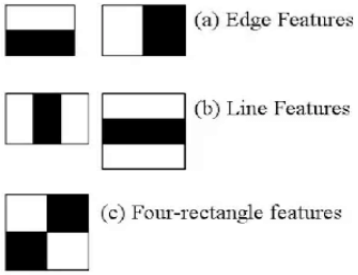
The key idea behind Haar cascade is that only a small number of pixels among the entire image is related to the object in concern. Therefore, it is essential to discard the irrelevant part of the image as quickly as possible. During the detection process, the Haar cascade scans the image at different scales and locations to eliminate irrelevant regions. The cascade structure, trained using the AdaBoost algorithm, enables an efficient, hierarchical evaluation of features, reducing the computational load and accelerating the detection speed.

Haar Cascade Algorithm

This involves Four Stages that include:

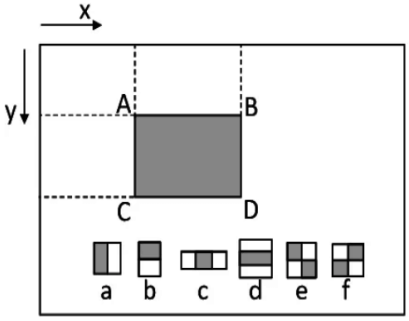
1. Haar Features Calculation
2. Integral Images Creation
3. Adaboost Usage
4. Cascading Classifiers Implementation

1. Haar Features Calculation: Gathering the Haar features is the first stage. Haar features are nothing but a calculation that happens on adjacent regions at a certain location in a separate detecting window. The calculation mainly includes adding the pixel intensities in every region and between the sum differences calculation.



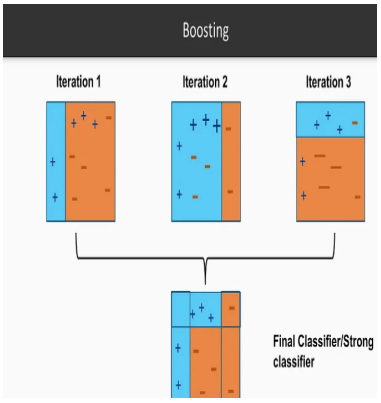
This is arduous in the case of large images because these integral images are used in which operations are reduced.

2. Integral Image Creation: Creating Integral Images reduces the calculation. Instead of calculating at every pixel, it creates the sub-rectangles, and the array references those sub-rectangles and calculates the Haar Features.



The only important features are those of an object, and mostly all the remaining Haar features are irrelevant in the case of object detection. But how do we choose from among the hundreds of thousands of Haar features the ones that best reflect an object?

3. Adaboost Training:

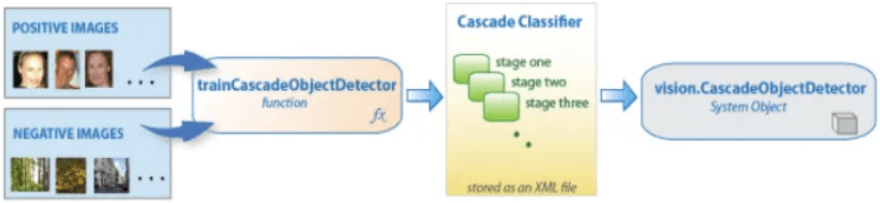


The "weak classifiers" are combined by Adaboost Training to produce a "strong classifier" that the object detection method can use. This essentially consists of selecting useful features and teaching classifiers how to use them.

By moving a window across the input image and computing the Haar characteristics for each part of the image, weak learners are created. This distinction stands in contrast to a threshold that can be trained to tell objects apart from non-objects. These are "weak classifiers," but an accurate strong classifier needs many Haar properties.

In the final step, weak learners might be combined with strong learners.

4. Cascading Classifiers Implementation:



Every sage at this point is actually a group of inexperienced students. Boosting trains weak learners, resulting in a highly accurate classifier from the average prediction of all weak learners.

It depends based upon the prediction. The classifier decides for indication of an object that was found positive or moved to the next region, i.e., negative. Because most windows do not contain anything of interest, stages are created to reject negative samples as quickly as feasible.

Because classifying an object as a non-object would significantly hurt your object detection system, having a low false negative rate is crucial.

The Haar Cascade Algorithm is used in several varieties of fields. Some of the applications are:

1. **Facial recognition:** Like how iPhone users use facial recognition like other electronic devices use the Haar Cascade Algorithm for security login to know about the validity of the user.
2. **Robotics:** These Robotics Machines can see the surroundings and perform tasks using Object Detection.
3. **Autonomous Vehicles:** Thes Autonomous Vehicles requires knowledge, and this Cascade algorithm can be able to identify the objects like pedestrians, traffic lights, etc., for safety purpose.
4. **Image Search and Object Recognition:** with the help of this Haar Cascade Algorithm, facial recognition expansion and different types of objects can be searched.
5. **Industrial Use:** Haar Cascade Algorithm allows machines to pick up and identifies objects.

Haar Cascade in OpenCV

Haar cascade is an algorithm, but first, you need to train a Haar cascade classifier before you can use it as an object detector.

- human face
- eye detection
- full body, upper body, or lower body of a human
- vehicle license plate

The pre-trained classifier is stored as an XML file. You can find the filename of the built-in classifiers from the GitHub link. To create a classifier, you must provide the path to this XML file. If you're using the one that shipped with OpenCV, you can use the following syntax:

```
# Load the Haar cascade for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

Usually a photo has multiple channels for the different colors (such as red, green, and blue). Haar cascade depends on pixel intensity only. Hence you should provide a single channel image, such as the grayscale version.

Using the Haar cascade classifier to detect objects is to use the method detectMultiScale(). It takes the following arguments:

1. **image:** This is the input image on which you want to perform object detection. It should be in grayscale format, or the "V" channel for an image in HSV channel format
2. **scaleFactor:** This parameter compensates for the fact that an object at different distances from the camera will appear at different sizes. It controls how much the image size is reduced at each image scale. It must be strictly greater than 1. A lower scaleFactor increases the detection time but also increases the chance of detection. Typical values range from 1.01 to 1.3.
3. **minNeighbors:** This parameter specifies how many neighbours each candidate object should have to retain it. Higher values result in fewer detections but with higher quality. Lower values may lead to more detections but with possible false positives. It's a trade-off between precision and recall.
4. **minSize**: This parameter sets the minimum object size. Objects smaller than this will be ignored. It's a tuple of the form (width, height).

A medium size resolution of 1920×1080 is used in this example. If you have a different resolution, you may need to tweak the arguments to detectMultiScale() below specifically the minSize.

Let's create a face detector and find the location of the faces of the pedestrians. The classifier is created using the pre-trained model haarcascade_frontalface_default.xml that shipped with OpenCV. The model file is located in the path pointed by cv2.data.haarcascades. Then we can use it to detect faces as bounding boxes:

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=4, minSize=(20, 20))
```

Feel free to adjust the parameters in your case. To illustrate the result, you can make use of OpenCV's function to draw on the original image,

```
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

Note that the cv2.rectangle() function asks for the coordinates of the opposite corners of a rectangular box, while the output of detectMultiScale() provides the coordinates of the top left corner and the width and height. The function above draws a blue box of two pixels wide on each face detected. Note that in OpenCV, images are presented in BGR channel order. Hence the pixel color (255, 0, 0) represents blue.

The result is as follows:

You can see that there are some false positives but overall, it provided a quite good result. You can adjust the parameters above to see how your result changes. The quality of the object detector using Haar cascade depends on how well it is trained to produce the model you read from the XML file.

```
import cv2
import sys
filename = 'image1.jpg'
# Load the Haar cascade for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Read the input image
img = cv2.imread(filename)
# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
# Perform face detection
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=4, minSize=(20, 20))

 # Draw rectangles around the detected faces
for (x, y, w, h) in faces:
   cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)

# Display the result
cv2.imshow('Face Detection', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Parameters of detectMultiScale:

Syntax: **detectMultiScale (image, objects, scaleFactor, minNeighbors, flags, minSize, maxSize)**

Example: **detectMultiScale (image, objects, scaleFactor = 1.1, minNeighbors = 3, flags = 0, minSize = new cv.Size(0, 0), maxSize = new cv.Size(0, 0))**

Option	Description
image	matrix of the type CV_8U containing an image where objects are detected.
objects	vector of rectangles where each rectangle contains the detected object. The rectangles may be partially outside the original image.
scaleFactor	parameter specifying how much the image size is reduced at each image scale.
minNeighbors	parameter specifying how many neighbors each candidate rectangle should have to retain it.
flags	parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.
minSize	minimum possible object size. Objects smaller than this are ignored.
maxSize	maximum possible object size. Objects larger than this are ignored. If maxSize == minSize model is evaluated on single scale.

Limitation of Haar Cascade

- High False Positives: Haar cascades often struggle with accurately distinguishing objects in cluttered or noisy backgrounds.
- Limited to Specific Objects: Pre-trained Haar cascades are effective only for specific objects.
- Sensitivity to Lighting: Performance decreases in low-light or high-contrast conditions.

Applications

- Real-time face tracking in lightweight systems.
- License plate detection in videos.
- Pedestrian detection for surveillance

Code for face detection

```
import cv2
import sys
filename = 'JPG file path'
# Load the Haar cascade for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Read the input image
img = cv2.imread(filename)
# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Perform face detection
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=4, minSize=(20, 20))
# Draw rectangles around the detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (100, 255, 50), 2)

# Display the result
cv2.imshow('Face Detection', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

code for video object detection

```
import cv2
# Load pre-trained Haar cascades for face and facial features
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +'haarcascade_eye.xml')
nose_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +'haarcascade_mcs_nose.xml')
mouth_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +'haarcascade_mcs_mouth.xml')
# Open video capture (0 for default webcam, or provide a video file)
filepath = r'file path of video '
cap = cv2.VideoCapture(filepath)
while True:
    ret, frame = cap.read()
    if not ret:
        break
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Detect faces in the image
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,minNeighbors=5, minSize=(30, 30))
    # Loop over the faces detected
```

```
    for (x, y, w, h) in faces:
        # Draw a rectangle around the face
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
        # Region of interest (ROI) for eyes, nose, mouth
        roi_gray = gray[y:y + h, x:x + w]
        roi_color = frame[y:y + h, x:x + w]

        # Detect eyes within the face region
        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)

        # Detect nose within the face region
        nose = nose_cascade.detectMultiScale(roi_gray)
        for (nx, ny, nw, nh) in nose:
            cv2.rectangle(roi_color, (nx, ny), (nx + nw, ny + nh), (0, 0, 255), 2)

        # Detect mouth within the face region
        mouth = mouth_cascade.detectMultiScale(roi_gray)
        for (mx, my, mw, mh) in mouth:
            cv2.rectangle(roi_color, (mx, my), (mx + mw, my + mh), (0, 255,255), 2)

        # Display the resulting frame
        cv2.imshow('Facial Features Detection', frame)
        # Exit when 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        # Release the video capture and close the window

cap.release()
cv2.destroyAllWindows()
```