Unit 1

What is Machine Learning?

Machine learning is a branch of artificial intelligence that enables algorithms to uncover hidden patterns within datasets. It allows them to predict new, similar data without explicit programming for each task. Machine learning finds applications in diverse fields such as image and speech recognition, natural language processing, recommendation systems, fraud detection, portfolio optimization, and automating tasks.

Machine learning?s impact extends to autonomous vehicles, drones, and robots, enhancing their adaptability in dynamic environments. This approach marks a breakthrough where machines learn from data examples to generate accurate outcomes, closely intertwined with data mining and data science.

Need for Machine Learning

Machine learning is important because it allows computers to learn from data and improve their performance on specific tasks without being explicitly programmed. This ability to learn from data and adapt to new situations makes machine learning particularly useful for tasks that involve large amounts of data, complex decision-making, and dynamic environments.

Here are some specific areas where machine learning is being used:

: Machine learning can be used to build predictive models that can help businesses make better decisions. For example, machine learning can be used to predict which customers are most likely to buy a particular product, or which patients are most likely to develop a certain disease.

: Machine learning is used to build systems that can understand and interpret human language. This is important for applications such as voice recognition, chatbots, and language translation.

: Machine learning is used to build systems that can recognize and interpret images and videos. This is important for applications such as self-driving cars, surveillance systems, and medical imaging.

: Machine learning can be used to detect fraudulent behavior in financial transactions, online advertising, and other areas.

: Machine learning can be used to build recommendation systems that suggest products, services, or content to users based on their past behavior and preferences.

Overall, machine learning has become an essential tool for many businesses and industries, as it enables them to make better use of data, improve their decision-making processes, and deliver more personalized experiences to their customers.

Difference between Machine Learning, Traditional Programming and Artificial Intelligence

How machine learning algorithms work?

A machine learning algorithm works by learning patterns and relationships from data to make predictions or decisions without being explicitly programmed for each task.

Here?s a simplified overview of how a typical machine learning algorithm works:

1. Data Collection

First, relevant data is collected or curated. This data could include examples, features, or attributes that are important for the task at hand, such as images, text, numerical data, etc.

2. Data Preprocessing

Before feeding the data into the algorithm, it often needs to be preprocessed. This step may involve cleaning the data (handling missing values, outliers), transforming the data (normalization, scaling), and splitting it into training and test sets.

3. Choosing a Model

Depending on the task (e.g., classification, regression, clustering), a suitable machine learning model is chosen. Examples include decision trees, neural networks, support vector machines, and more advanced models like deep learning architectures.

4. Training the Model

The selected model is trained using the training data. During training, the algorithm learns patterns and relationships in the data. This involves adjusting model parameters iteratively to minimize the difference between predicted outputs and actual outputs (labels or targets) in the training data.

5. Evaluating the Model

Once trained, the model is evaluated using the test data to assess its performance. Metrics such as accuracy, precision, recall, or mean squared error are used to evaluate how well the model generalizes to new, unseen data.

6. Fine-tuning

Models may be fine-tuned by adjusting hyperparameters (parameters that are not directly learned during training, like learning rate or number of hidden layers in a neural network) to improve performance.

7. Prediction or Inference

Finally, the trained model is used to make predictions or decisions on new data. This process involves applying the learned patterns to new inputs to generate outputs, such as class labels in classification tasks or numerical values in regression tasks.

Machine Learning Lifecycle

The includes:

Defining the Problem: Clearly identify the real-world problem to be solved.

Data Collection: Gather necessary data from various sources.

Data Cleaning and Preprocessing: Resolve data quality issues and prepare the data for analysis.

Exploratory Data Analysis (EDA): Analyze data to identify patterns, outliers, and trends.

Feature Engineering and Selection: Enhance data features and select relevant ones to improve model performance.

Model Selection: Choose suitable models based on the problem type and data characteristics.

Model Training: Train the model using a split of training and validation datasets.

Model Evaluation and Tuning: Assess and optimize the model using relevant metrics.

Model Deployment: Implement the model in a production environment for real-time predictions.

Model Monitoring and Maintenance: Regularly check and update the model to maintain accuracy.

Machine Learning Life Cycle

Types of Machine Learning

Machine learning is the branch of that focuses on developing models and algorithms that let computers learn from data and improve from previous experience without being explicitly programmed for every task.In simple words, ML teaches the systems to think and understand like humans by learning from the data.

In this article, we will explore the various types of that are important for future requirements. Machine learning is generally a training system to learn from past experiences and improve performance over time. helps to predict massive amounts of data. It helps to deliver fast and accurate results to get profitable opportunities.

Types of Machine Learning

There are several types of machine learning, each with special characteristics and applications. Some of the main types of machine learning algorithms are as follows:

Supervised Machine Learning

Unsupervised Machine Learning

Reinforcement Learning

Additionally, there is a more specific category called semi-supervised learning, which combines elements of both supervised and unsupervised learning.

Types of Machine Learning

1. Supervised Machine Learning

is defined as when a model gets trained on a ?Labelled Dataset?. Labelled datasets have both input and output parameters. In Supervised Learning algorithms learn to map points between inputs and correct outputs. It has both training and validation datasets labelled.

Supervised Learning

Let?s understand it with the help of an example.

Example: Consider a scenario where you have to build an image classifier to differentiate between cats and dogs. If you feed the datasets of dogs and cats labelled images to the algorithm, the machine will learn to classify between a dog or a cat from these labeled images. When we input new dog or cat images that it has never seen before, it will use the learned algorithms and predict whether it is a dog or a cat. This is how supervised learning works, and this is particularly an image classification.

There are two main categories of supervised learning that are mentioned below:

Classification

deals with predicting categorical target variables, which represent discrete classes or labels. For instance, classifying emails as spam or not spam, or predicting whether a patient has a high risk of heart disease. Classification algorithms learn to map the input features to one of the predefined classes.

Here are some classification algorithms:

Regression

, on the other hand, deals with predicting continuous target variables, which represent numerical values. For example, predicting the price of a house based on its size, location, and amenities, or forecasting the sales of a product. Regression algorithms learn to map the input features to a continuous numerical value.

Here are some regression algorithms:

Advantages of Supervised Machine Learning

Supervised Learning models can have high accuracy as they are trained on labelled data.

The process of decision-making in supervised learning models is often interpretable.

It can often be used in pre-trained models which saves time and resources when developing new models from scratch.

Disadvantages of Supervised Machine Learning

It has limitations in knowing patterns and may struggle with unseen or unexpected patterns that are not present in the training data.

It can be time-consuming and costly as it relies on labeled data only.

It may lead to poor generalizations based on new data.

Applications of Supervised Learning

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

Image classification: Identify objects, faces, and other features in images.

Natural language processing: Extract information from text, such as sentiment, entities, and relationships.

Speech recognition: Convert spoken language into text.

Recommendation systems: Make personalized recommendations to users.

Predictive analytics: Predict outcomes, such as sales, customer churn, and stock prices.

Medical diagnosis: Detect diseases and other medical conditions.

Fraud detection: Identify fraudulent transactions.

Autonomous vehicles: Recognize and respond to objects in the environment.

Email spam detection: Classify emails as spam or not spam.

Quality control in manufacturing: Inspect products for defects.

Credit scoring: Assess the risk of a borrower defaulting on a loan.

Gaming: Recognize characters, analyze player behavior, and create NPCs.

Customer support: Automate customer support tasks.

Weather forecasting: Make predictions for temperature, precipitation, and other meteorological parameters.

Sports analytics: Analyze player performance, make game predictions, and optimize strategies.

2. Unsupervised Machine Learning

Unsupervised learning is a type of machine learning technique in which an algorithm discovers patterns and relationships using unlabeled data. Unlike supervised learning, unsupervised learning doesn?t involve providing the algorithm with labeled target outputs. The primary goal of Unsupervised learning is often to discover hidden patterns, similarities, or clusters within the data, which can then be used for various purposes, such as data exploration, visualization, dimensionality reduction, and more.

Unsupervised Learning

Let?s understand it with the help of an example.

Example: Consider that you have a dataset that contains information about the purchases you made from the shop. Through clustering, the algorithm can group the same purchasing behavior among you and other customers, which reveals potential customers without predefined labels. This type of information can help businesses get target customers as well as identify outliers.

There are two main categories of unsupervised learning that are mentioned below:

Clustering

is the process of grouping data points into clusters based on their similarity. This technique is useful for identifying patterns and relationships in data without the need for labeled examples.

Here are some clustering algorithms:

Association

ing is a technique for discovering relationships between items in a dataset. It identifies rules that indicate the presence of one item implies the presence of another item with a specific probability.

Here are some association rule learning algorithms:

Advantages of Unsupervised Machine Learning

It helps to discover hidden patterns and various relationships between the data.

Used for tasks such as customer segmentation, anomaly detection, and data exploration.

It does not require labeled data and reduces the effort of data labeling.

Disadvantages of Unsupervised Machine Learning

Without using labels, it may be difficult to predict the quality of the model?s output.

Cluster Interpretability may not be clear and may not have meaningful interpretations.

It has techniques such as and that can be used to extract meaningful features from raw data.

Applications of Unsupervised Learning

Here are some common applications of unsupervised learning:

Clustering: Group similar data points into clusters.

Anomaly detection: Identify outliers or anomalies in data.

Dimensionality reduction: Reduce the dimensionality of data while preserving its essential information.

Recommendation systems: Suggest products, movies, or content to users based on their historical behavior or preferences.

Topic modeling: Discover latent topics within a collection of documents.

Density estimation: Estimate the probability density function of data.

Image and video compression: Reduce the amount of storage required for multimedia content.

Data preprocessing: Help with data preprocessing tasks such as data cleaning, imputation of missing values, and data scaling.

Market basket analysis: Discover associations between products.

Genomic data analysis: Identify patterns or group genes with similar expression profiles.

Image segmentation: Segment images into meaningful regions.

Community detection in social networks: Identify communities or groups of individuals with similar interests or connections.

Customer behavior analysis: Uncover patterns and insights for better marketing and product recommendations.

Content recommendation: Classify and tag content to make it easier to recommend similar items to users.

Exploratory data analysis (EDA): Explore data and gain insights before defining specific tasks.

3. Reinforcement Machine Learning

algorithm is a learning method that interacts with the environment by producing actions and discovering errors. Trial, error, and delay are the most relevant characteristics of reinforcement learning. In this technique, the model keeps on increasing its performance using Reward Feedback to learn the behavior or pattern. These algorithms are specific to a particular problem e.g. Google Self Driving car, AlphaGo where a bot competes with humans and even itself to get better and better performers in Go Game. Each time we feed in data, they learn and add the data to their knowledge which is training data. So, the more it learns the better it gets trained and hence experienced.

Here are some of most common reinforcement learning algorithms:

Q-learning is a model-free RL algorithm that learns a Q-function, which maps states to actions. The Q-function estimates the expected reward of taking a particular action in a given state.

SARSA is another model-free RL algorithm that learns a Q-function. However, unlike Q-learning, SARSA updates the Q-function for the action that was actually taken, rather than the optimal action.

: Deep Q-learning is a combination of Q-learning and deep learning. Deep Q-learning uses a neural network to represent the Q-function, which allows it to learn complex relationships between states and actions.

Reinforcement Machine Learning

Let?s understand it with the help of examples.

Example: Consider that you are training an agent to play a game like chess. The agent explores different moves and receives positive or negative feedback based on the outcome. Reinforcement Learning also finds applications in which they learn to perform tasks by interacting with their surroundings.

Types of Reinforcement Machine Learning

There are two main types of reinforcement learning:

Positive reinforcement

Rewards the agent for taking a desired action.

Encourages the agent to repeat the behavior.

Examples: Giving a treat to a dog for sitting, providing a point in a game for a correct answer.

Negative reinforcement

Removes an undesirable stimulus to encourage a desired behavior.

Discourages the agent from repeating the behavior.

Examples: Turning off a loud buzzer when a lever is pressed, avoiding a penalty by completing a task.

Advantages of Reinforcement Machine Learning

It has autonomous decision-making that is well-suited for tasks and that can learn to make a sequence of decisions, like robotics and game-playing.

This technique is preferred to achieve long-term results that are very difficult to achieve.

It is used to solve a complex problems that cannot be solved by conventional techniques.

Disadvantages of Reinforcement Machine Learning

Training Reinforcement Learning agents can be computationally expensive and time-consuming.

Reinforcement learning is not preferable to solving simple problems.

It needs a lot of data and a lot of computation, which makes it impractical and costly.

Applications of Reinforcement Machine Learning

Here are some applications of reinforcement learning:

Game Playing: RL can teach agents to play games, even complex ones.

Robotics: RL can teach robots to perform tasks autonomously.

Autonomous Vehicles: RL can help self-driving cars navigate and make decisions.

Recommendation Systems: RL can enhance recommendation algorithms by learning user preferences.

Healthcare: RL can be used to optimize treatment plans and drug discovery.

Natural Language Processing (NLP): RL can be used in dialogue systems and chatbots.

Finance and Trading: RL can be used for algorithmic trading.

Supply Chain and Inventory Management: RL can be used to optimize supply chain operations.

Energy Management: RL can be used to optimize energy consumption.

Game AI: RL can be used to create more intelligent and adaptive NPCs in video games.

Adaptive Personal Assistants: RL can be used to improve personal assistants.

Virtual Reality (VR) and Augmented Reality (AR): RL can be used to create immersive and interactive experiences.

Industrial Control: RL can be used to optimize industrial processes.

Education: RL can be used to create adaptive learning systems.

Agriculture: RL can be used to optimize agricultural operations.

Semi-Supervised Learning: Supervised + Unsupervised Learning

is a machine learning algorithm that works between the supervised and unsupervised learning so it uses both labelled and unlabelled data. It?s particularly useful when obtaining labeled data is costly, time-consuming, or resource-intensive. This approach is useful when the dataset is expensive and time-consuming. Semi-supervised learning is chosen when labeled data requires skills and relevant resources in order to train or learn from it.

We use these techniques when we are dealing with data that is a little bit labeled and the rest large portion of it is unlabeled. We can use the unsupervised techniques to predict labels and then feed these labels to supervised techniques. This technique is mostly applicable in the case of image data sets where usually all images are not labeled.

Semi-Supervised Learning

Let?s understand it with the help of an example.

Example: Consider that we are building a language translation model, having labeled translations for every sentence pair can be resources intensive. It allows the models to learn from labeled and unlabeled sentence pairs, making them more accurate. This technique has led to significant improvements in the quality of machine translation services.

Types of Semi-Supervised Learning Methods

There are a number of different semi-supervised learning methods each with its own characteristics. Some of the most common ones include:

Graph-based semi-supervised learning: This approach uses a graph to represent the relationships between the data points. The graph is then used to propagate labels from the labeled data points to the unlabeled data points.

Label propagation: This approach iteratively propagates labels from the labeled data points to the unlabeled data points, based on the similarities between the data points.

Co-training: This approach trains two different machine learning models on different subsets of the unlabeled data. The two models are then used to label each other?s predictions.

Self-training: This approach trains a machine learning model on the labeled data and then uses the model to predict labels for the unlabeled data. The model is then retrained on the labeled data and the predicted labels for the unlabeled data.

: GANs are a type of deep learning algorithm that can be used to generate synthetic data. GANs can be used to generate unlabeled data for semi-supervised learning by training two neural networks, a generator and a discriminator.

Advantages of Semi- Supervised Machine Learning

It leads to better generalization as compared to supervised learning, as it takes both labeled and unlabeled data.

Can be applied to a wide range of data.

Disadvantages of Semi- Supervised Machine Learning

Semi-supervised methods can be more complex to implement compared to other approaches.

It still requires some labeled data that might not always be available or easy to obtain.

The unlabeled data can impact the model performance accordingly.

Applications of Semi-Supervised Learning

Here are some common applications of semi-supervised learning:

Image Classification and Object Recognition: Improve the accuracy of models by combining a small set of labeled images with a larger set of unlabeled images.

Natural Language Processing (NLP): Enhance the performance of language models and classifiers by combining a small set of labeled text data with a vast amount of unlabeled text.

Speech Recognition: Improve the accuracy of speech recognition by leveraging a limited amount of transcribed speech data and a more extensive set of unlabeled audio.

Recommendation Systems: Improve the accuracy of personalized recommendations by supplementing a sparse set of user-item interactions (labeled data) with a wealth of unlabeled user behavior data.

Healthcare and Medical Imaging: Enhance medical image analysis by utilizing a small set of labeled medical images alongside a larger set of unlabeled images.

Conclusion

In conclusion, each type of machine learning serves its own purpose and contributes to the overall role in development of enhanced data prediction capabilities, and it has the potential to change various industries like . It helps deal with massive data production and management of the datasets.

Types of Machine Learning ? FAQs

1. What are the challenges faced in supervised learning?

Some of the challenges faced in supervised learning mainly include addressing class imbalances, high-quality labeled data, and avoiding overfitting where models perform badly on real-time data.

2. Where can we apply supervised learning?

Supervised learning is commonly used for tasks like analysing spam emails, image recognition, and sentiment analysis.

3. What does the future of machine learning outlook look like?

Machine learning as a future outlook may work in areas like weather or climate analysis, healthcare systems, and autonomous modelling.

4. What are the different types of machine learning?

There are three main types of machine learning:

Supervised learning

Unsupervised learning

Reinforcement learning

5. What are the most common machine learning algorithms?

Some of the most common machine learning algorithms include:

Linear regression

Logistic regression

Support vector machines (SVMs)

K-nearest neighbors (KNN)

Decision trees

Random forests

Artificial Neural networks

Linear Regression in Machine Learning

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:

Mathematically, we can represent a linear regression as:

y= a0+a1x+ ?

Here,

Y= Dependent Variable (Target Variable)

X= Independent Variable (predictor Variable)

a0= intercept of the line (Gives an additional degree of freedom)

a1 = Linear regression coefficient (scale factor to each input value).

? = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

Simple Linear Regression:

If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

Multiple Linear regression:

If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

A linear line showing the relationship between the dependent and independent variables is called a regression line. A regression line can show two types of relationship:

Positive Linear Relationship:

If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.

Negative Linear Relationship:

If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a0, a1) gives a different line of regression, so we need to calculate the best values for a0 and a1 to find the best fit line, so to calculate this we use cost function.

Cost function-

The different values for weights or coefficient of lines (a0, a1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.

Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.

We can use the cost function to find the accuracy of the mapping function, which maps the input variable to the output variable. This mapping function is also known as Hypothesis function.

For Linear Regression, we use the Mean Squared Error (MSE) cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

Where,

N=Total number of observation

Yi = Actual value

(a1xi+a0)= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent:

Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.

A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.

It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called optimization. It can be achieved by below method:

1. R-squared method:

R-squared is a statistical method that determines the goodness of fit.

It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.

The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.

It is also called a coefficient of determination, or coefficient of multiple determination for multiple regression.

It can be calculated from the below formula:

Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

Linear relationship between the features and target:

Linear regression assumes the linear relationship between the dependent and independent variables.

Small or no multicollinearity between the features:

Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.

Homoscedasticity Assumption:

Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.

Normal distribution of error terms:

Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.

It can be checked using the q-q plot. If the plot shows a straight line without any deviation, which means the error is normally distributed.

No autocorrelations:

The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

Simple Linear Regression in Machine Learning

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

The key point in Simple Linear Regression is that the dependent variable must be a continuous/real value. However, the independent variable can be measured on continuous or categorical values.

Simple Linear regression algorithm has mainly two objectives:

Model the relationship between the two variables. Such as the relationship between Income and expenditure, experience and Salary, etc.

Forecasting new observations. Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

Simple Linear Regression Model:

The Simple Linear Regression model can be represented using the below equation:

y= a0+a1x+ ?

Where,

a0= It is the intercept of the Regression line (can be obtained putting x=0)

a1= It is the slope of the regression line, which tells whether the line is increasing or decreasing.

? = The error term. (For a good model it will be negligible)

Implementation of Simple Linear Regression Algorithm using Python

Problem Statement example for Simple Linear Regression:

Here we are taking a dataset that has two variables: salary (dependent variable) and experience (Independent variable). The goals of this problem is:

We want to find out if there is any correlation between these two variables

We will find the best fit line for the dataset.

How the dependent variable is changing by changing the independent variable.

In this section, we will create a Simple Linear Regression model to find out the best fitting line for representing the relationship between these two variables.

To implement the Simple Linear regression model in machine learning using Python, we need to follow the below steps:

Step-1: Data Pre-processing

The first step for creating the Simple Linear Regression model is . We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

```
import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd
```

Next, we will load the dataset into our code:

```
data_set= pd.read_csv('Salary_Data.csv')
```

By executing the above line of code (ctrl+ENTER), we can read the dataset on our Spyder IDE screen by clicking on the variable explorer option.

The above output shows the dataset, which has two variables: Salary and Experience.

Note: In Spyder IDE, the folder containing the code file must be saved as a working directory, and the dataset or csv file should be in the same folder.

After that, we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Below is code for it:

```
x= data_set.iloc[:, :-1].values

y= data_set.iloc[:, 1].values
```

In the above lines of code, for x variable, we have taken -1 value since we want to remove the last column from the dataset. For y variable, we have taken 1 value as a parameter, since we want to extract the second column and indexing starts from the zero.

By executing the above line of code, we will get the output for X and Y variable as:

In the above output image, we can see the X (independent) variable and Y (dependent) variable has been extracted from the given dataset.

Next, we will split both variables into the test set and training set. We have 30 observations, so we will take 20 observations for the training set and 10 observations for the test set. We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset. The code for this is given below:

```
# Splitting the dataset into training 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= 1/3, random_state=0)
```

By executing the above code, we will get x-test, x-train and y-test, y-train dataset. Consider the below images:

Test-dataset:

Training Dataset:

For simple linear Regression, we will not use Feature Scaling. Because Python libraries take care of it for some cases, so we don't need to perform it here. Now, our dataset is well prepared to work on it and we are going to start building a Simple Linear Regression model for the given problem.

Step-2: Fitting the Simple Linear Regression to the Training Set:

Now the second step is to fit our model to the training dataset. To do so, we will import the LinearRegression class of the linear_model library from the scikit learn. After importing the class, we are going to create an object of the class named as a regressor. The code for this is given below:

```
#Fitting the Simple Linear Regression model to the training dataset
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

In the above code, we have used a fit() method to fit our Simple Linear Regression object to the training set. In the fit() function, we have passed the x_train and y_train, which is our training dataset for the dependent and an independent variable. We have fitted our regressor object to the training set so that the model can easily learn the correlations between the predictor and target variables. After executing the above lines of code, we will get the below output.

Output:

```
Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Step: 3. Prediction of test set result:

dependent (salary) and an independent variable (Experience). So, now, our model is ready to predict the output for the new observations. In this step, we will provide the test dataset (new observations) to the model to check whether it can predict the correct output or not.

We will create a prediction vector y_pred, and x_pred, which will contain predictions of test dataset, and prediction of training set respectively.

#Prediction of Test and Training set result

y_pred= regressor.predict(x_test)

x_pred= regressor.predict(x_train)

On executing the above lines of code, two variables named y_pred and x_pred will generate in the variable explorer options that contain salary predictions for the training set and test set.

Output:

You can check the variable by clicking on the variable explorer option in the IDE, and also compare the result by comparing values from y_pred and y_test. By comparing these values, we can check how good our model is performing.

Step: 4. visualizing the Training set results:

Now in this step, we will visualize the training set result. To do so, we will use the scatter() function of the pyplot library, which we have already imported in the pre-processing step. The scatter () function will create a scatter plot of observations.

In the x-axis, we will plot the Years of Experience of employees and on the y-axis, salary of employees. In the function, we will pass the real values of training set, which means a year of experience x_train, training set of Salaries y_train, and color of the observations. Here we are taking a green color for the observation, but it can be any color as per the choice.

Now, we need to plot the regression line, so for this, we will use the plot() function of the pyplot library. In this function, we will pass the years of experience for training set, predicted salary for training set x_pred, and color of the line.

Next, we will give the title for the plot. So here, we will use the title() function of the pyplot library and pass the name ("Salary vs Experience (Training Dataset)".

After that, we will assign labels for x-axis and y-axis using xlabel() and ylabel() function.

Finally, we will represent all above things in a graph using show(). The code is given below:

```
mtp.scatter(x_train, y_train, color="green")

mtp.plot(x_train, x_pred, color="red")

mtp.title("Salary vs Experience (Training Dataset)")

mtp.xlabel("Years of Experience")

mtp.ylabel("Salary(In Rupees)")

mtp.show()
```

Output:

By executing the above lines of code, we will get the below graph plot as an output.

In the above plot, we can see the real values observations in green dots and predicted values are covered by the red regression line. The regression line shows a correlation between the dependent and independent variable.

The good fit of the line can be observed by calculating the difference between actual values and predicted values. But as we can see in the above plot, most of the observations are close to the regression line, hence our model is good for the training set.

Step: 5. visualizing the Test set results:

In the previous step, we have visualized the performance of our model on the training set. Now, we will do the same for the Test set. The complete code will remain the same as the above code, except in this, we will use x_test, and y_test instead of x_train and y_train.

Here we are also changing the color of observations and regression line to differentiate between the two plots, but it is optional.

```
#visualizing the Test set results

mtp.scatter(x_test, y_test, color="blue")

mtp.plot(x_train, x_pred, color="red")

mtp.title("Salary vs Experience (Test Dataset)")

mtp.xlabel("Years of Experience")

mtp.ylabel("Salary(In Rupees)")

mtp.show()
```

Output:

By executing the above line of code, we will get the output as:

In the above plot, there are observations given by the blue color, and prediction is given by the red regression line. As we can see, most of the observations are close to the regression line, hence we can say our Simple Linear Regression is a good model and able to make good predictions.

Multiple Linear Regression

In the previous topic, we have learned about Simple Linear Regression, where a single Independent/Predictor(X) variable is used to model the response variable (Y). But there may be various cases in which the response variable is affected by more than one predictor variable; for such cases, the Multiple Linear Regression algorithm is used.

Moreover, Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable. We can define it as:

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Example:

Prediction of CO2 emission based on engine size and number of cylinders in a car.

Some key points about MLR:

For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.

Each feature variable must model the linear relationship with the dependent variable.

MLR tries to fit a regression line through a multidimensional space of data-points.

MLR equation:

In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables x1, x2, x3, ...,xn. Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

Y= b<sub>0</sub>+b<sub>1</sub>x<sub>1</sub>+ b<sub>2</sub>x<sub>2</sub>+ b<sub>3</su

b>x<sub>3</sub>+...... bnxn   .... (a)

Where,

Y= Output/Response variable

b0, b1, b2, b3 , bn....= Coefficients of the model.

x1, x2, x3, x4,...= Various Independent/feature variable

Assumptions for Multiple Linear Regression:

A linear relationship should exist between the Target and predictor variables.

The regression residuals must be normally distributed.

MLR assumes little or no multicollinearity (correlation between the independent variable) in data.

Implementation of Multiple Linear Regression model using Python:

To implement MLR using Python, we have below problem:

Problem Description:

We have a dataset of 50 start-up companies. This dataset contains five main information: R&D Spend, Administration Spend, Marketing Spend, State, and Profit for a financial year. Our goal is to create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.

Since we need to find the Profit, so it is the dependent variable, and the other four variables are independent variables. Below are the main steps of deploying the MLR model:

Data Pre-processing Steps

Fitting the MLR model to the training set

Predicting the result of the test set

Step-1: Data Pre-processing Step:

The very first step is , which we have already discussed in this tutorial. This process contains the below steps:

Importing libraries: Firstly we will import the library which will help in building the model. Below is the code for it:

# importing libraries

```
import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd
```

Importing dataset: Now we will import the dataset(50_CompList), which contains all the variables. Below is the code for it:

```
#importing datasets

data_set= pd.read_csv('50_CompList.csv')
```

Output: We will get the dataset as:

In above output, we can clearly see that there are five variables, in which four variables are continuous and one is categorical variable.

Extracting dependent and independent Variables:

```
#Extracting Independent and dependent Variable

x= data_set.iloc[:, :-1].values

y= data_set.iloc[:, 4].values
```

Output:

```
Out[5]:

array([[165349.2, 136897.8, 471784.1, 'New York'],

[162597.7, 151377.59, 443898.53, 'California'],

[153441.51, 101145.55, 407934.54, 'Florida'],

[144372.41, 118671.85, 383199.62, 'New York'],

[142107.34, 91391.77, 366168.42, 'Florida'],

[131876.9, 99814.71, 362861.36, 'New York'],

[134615.46, 147198.87, 127716.82, 'California'],

[130298.13, 145530.06, 323876.68, 'Florida'],

[120542.52, 148718.95, 311613.29, 'New York'],

[123334.88, 108679.17, 304981.62, 'California'],

[101913.08, 110594.11, 229160.95, 'Florida'],
[100671.96, 91790.61, 249744.55, 'California'],

[93863.75, 127320.38, 249839.44, 'Florida'],

[91992.39, 135495.07, 252664.93, 'California'],

[119943.24, 156547.42, 256512.92, 'Florida'],

[114523.61, 122616.84, 261776.23, 'New York'],

[78013.11, 121597.55, 264346.06, 'California'],

[94657.16, 145077.58, 282574.31, 'New York'],

[91749.16, 114175.79, 294919.57, 'Florida'],

[86419.7, 153514.11, 0.0, 'New York'],

[76253.86, 113867.3, 298664.47, 'California'],

[78389.47, 153773.43, 299737.29, 'New York'],

[73994.56, 122782.75, 303319.26, 'Florida'],

[67532.53, 105751.03, 304768.73, 'Florida'],

[77044.01, 99281.34, 140574.81, 'New York'],

[64664.71, 139553.16, 137962.62, 'California'],

[75328.87, 144135.98, 134050.07, 'Florida'],

[72107.6, 127864.55, 353183.81, 'New York'],

[66051.52, 182645.56, 118148.2, 'Florida'],

[65605.48, 153032.06, 107138.38, 'New York'],

[61994.48, 115641.28, 91131.24, 'Florida'],

[61136.38, 152701.92, 88218.23, 'New York'],

[63408.86, 129219.61, 46085.25, 'California'],

[55493.95, 103057.49, 214634.81, 'Florida'],

[46426.07, 157693.92, 210797.67, 'California'],

[46014.02, 85047.44, 205517.64, 'New York'],

[28663.76, 127056.21, 201126.82, 'Florida'],

[44069.95, 51283.14, 197029.42, 'California'],
[20229.59, 65947.93, 185265.1, 'New York'],
[38558.51, 82982.09, 174999.3, 'California'],
[28754.33, 118546.05, 172795.67, 'California'],
[27892.92, 84710.77, 164470.71, 'Florida'],
[23640.93, 96189.63, 148001.11, 'California'],
[15505.73, 127382.3, 35534.17, 'New York'],
[22177.74, 154806.14, 28334.72, 'California'],
[1000.23, 124153.04, 1903.93, 'New York'],
[1315.46, 115816.21, 297114.46, 'Florida'],
[0.0, 135426.92, 0.0, 'California'],
[542.05, 51743.15, 0.0, 'New York'],
[0.0, 116983.8, 45173.06, 'California']], dtype=object)
```

As we can see in the above output, the last column contains categorical variables which are not suitable to apply directly for fitting the model. So we need to encode this variable.

Encoding Dummy Variables:

As we have one categorical variable (State), which cannot be directly applied to the model, so we will encode it. To encode the categorical variable into numbers, we will use the LabelEncoder class. But it is not sufficient because it still has some relational order, which may create a wrong model. So in order to remove this problem, we will use OneHotEncoder, which will create the dummy variables. Below is code for it:

```
#Catgorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_x= LabelEncoder()
x[:, 3]= labelencoder_x.fit_transform(x[:,3])
onehotencoder= OneHotEncoder(categorical_features= [3])
x= onehotencoder.fit_transform(x).toarray()
```

Here we are only encoding one independent variable, which is state as other variables are continuous.

Output:

As we can see in the above output, the state column has been converted into dummy variables (0 and 1). Here each dummy variable column is corresponding to the one State. We can check by comparing it with the original dataset. The first column corresponds to the California State, the second column corresponds to the Florida State, and the third column corresponds to the New York State.

Note: We should not use all the dummy variables at the same time, so it must be 1 less than the total number of dummy variables, else it will create a dummy variable trap.

Now, we are writing a single line of code just to avoid the dummy variable trap:

```
#avoiding the dummy variable trap:
x = x[:, 1:]
```

If we do not remove the first dummy variable, then it may introduce multicollinearity in the model.

As we can see in the above output image, the first column has been removed.

Now we will split the dataset into training and test set. The code for this is given below:

```
# Splitting the dataset into training 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= 0.2, random_state=0)
```

The above code will split our dataset into a training set and test set.

Output: The above code will split the dataset into training set and test set. You can check the output by clicking on the variable explorer option given in Spyder IDE. The test set and training set will look like the below image:

Test set:

Training set:

Note: In MLR, we will not do feature scaling as it is taken care by the library, so we don't need to do it manually.

Step: 2- Fitting our MLR model to the Training set:

Now, we have well prepared our dataset in order to provide training, which means we will fit our regression model to the training set. It will be similar to as we did in model. The code for this will be:

```
#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

Output:

```
Out[9]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Now, we have successfully trained our model using the training dataset. In the next step, we will test the performance of the model using the test dataset.

Step: 3- Prediction of Test set results:

The last step for our model is checking the performance of the model. We will do it by predicting the test set result. For prediction, we will create a y_pred vector. Below is the code for it:

```
#Predicting the Test set result;
y_pred= regressor.predict(x_test)
```

By executing the above lines of code, a new vector will be generated under the variable explorer option. We can test our model by comparing the predicted values and test set values.

Output:

In the above output, we have predicted result set and test set. We can check model performance by comparing these two value index by index. For example, the first index has a predicted value of 103015$ profit and test/real value of 103282$ profit. The difference is only of 267$, which is a good prediction, so, finally, our model is completed here.

We can also check the score for training dataset and test dataset. Below is the code for it:

```
print('Train Score: ', regressor.score(x_train, y_train))
print('Test Score: ', regressor.score(x_test, y_test))
```

Output: The score is:

```
Train Score: 0.9501847627493607
```

Test Score: 0.9347068473282446

The above score tells that our model is 95% accurate with the training dataset and 93% accurate with the test dataset.

Note: In the next topic, we will see how we can improve the performance of the model using the Backward Elimination process.

Applications of Multiple Linear Regression:

There are mainly two applications of Multiple Linear Regression:

Effectiveness of Independent variable on prediction:

Predicting the impact of changes:

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 color, 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 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:

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:

Convert the given dataset into frequency tables.

Generate Likelihood table by finding the probabilities of given features.

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:

Frequency table for the Weather Conditions:

Likelihood table weather condition:

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.

1) 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 . 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)
```

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:

2) 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)
```

3) 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.

4) 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.

5) 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 = 0.0
1),
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 (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.

6) 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 = 0.0
1),
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.

MCQ Exercise on Naïve Bayes Classifier Algorithm

1. Which of the following assumptions is made by the Naive Bayes classifier?

All features are independent of each other.

All features are dependent on each other.

Features are independent only if they have a Gaussian distribution.

Features are dependent only if they have a Gaussian distribution.

Hide Answer Workspace

Answer:

a) All features are independent of each other.

Explanation:

The Naive Bayes classifier assumes that all features are independent of each other given the class label, which is why it's called "naive."

2. Assertion (A): Naive Bayes Classifier assumes that the features are independent of each other.

Reason (R): The Naive Bayes Classifier applies Bayes' theorem with strong independence assumptions between the features.

Options:

Both A and R are true, and R is the correct explanation for A.

Both A and R are true, but R is not the correct explanation for A.

A is true, but R is false.

A is false, but R is true.

Hide Answer Workspace

Answer:

a) Both A and R are true, and R is the correct explanation for A.

Explanation:

The Naive Bayes Classifier is based on Bayes' theorem and assumes that the presence of a particular feature in a class is independent of the presence of any other feature. This strong assumption simplifies the computation and is the reason behind the "naive" in the classifier's name.

3. In text classification using Naive Bayes, what is the purpose of Laplace smoothing?

To handle missing values in the text.

To prevent zero probabilities for unseen words.

To improve the computational efficiency of the classifier.

To normalize the probabilities of words in the text.

Hide Answer Workspace

Answer:

b) To prevent zero probabilities for unseen words.

Explanation:

Laplace smoothing is used to avoid zero probabilities for words that are not present in the training dataset.

4. Which of the following types of Naive Bayes classifiers is most suitable for text classification tasks?

Gaussian Naive Bayes.

Multinomial Naive Bayes.

Bernoulli Naive Bayes.

Complement Naive Bayes.

Hide Answer Workspace

Answer:

b) Multinomial Naive Bayes.

Explanation:

Multinomial Naive Bayes is commonly used for text classification where features are typically word frequencies

5. Match the Following:

Gaussian Naive Bayes

Multinomial Naive Bayes

Bernoulli Naive Bayes

Used for binary/Boolean features

Used for continuous features

Used for discrete features like word counts in text classification

Options:

A-2, B-3, C-1

A-1, B-2, C-3

A-3, B-1, C-2

A-2, B-1, C-3

Hide Answer Workspace

Answer:

a) A-2, B-3, C-1

Explanation:

Gaussian Naive Bayes (A-2): Used for continuous features, assumes features follow a normal distribution.

Multinomial Naive Bayes (B-3): Used for discrete features like word counts in text classification.

Bernoulli Naive Bayes (C-1): Used for binary/Boolean features.

Regression vs. Classification in Machine Learning

Regression and Classification algorithms are Supervised Learning algorithms. Both the algorithms are used for prediction in Machine learning and work with the labeled datasets. But the difference between both is how they are used for different machine learning problems.

The main difference between Regression and Classification algorithms that Regression algorithms are used to predict the continuous values such as price, salary, age, etc. and Classification algorithms are used to predict/Classify the discrete values such as Male or Female, True or False, Spam or Not Spam, etc.

Consider the below diagram:

Classification:

Classification is a process of finding a function which helps in dividing the dataset into classes based on different parameters. In Classification, a computer program is trained on the training dataset and based on that training, it categorizes the data into different classes.

The task of the classification algorithm is to find the mapping function to map the input(x) to the discrete output(y).

Example: The best example to understand the Classification problem is Email Spam Detection. The model is trained on the basis of millions of emails on different parameters, and whenever it receives a new email, it identifies whether the email is spam or not. If the email is spam, then it is moved to the Spam folder.

Types of ML Classification Algorithms:

Classification Algorithms can be further divided into the following types:

Logistic Regression

K-Nearest Neighbours

Support Vector Machines

Kernel SVM

Na?ve Bayes

Decision Tree Classification

Random Forest Classification

Regression:

Regression is a process of finding the correlations between dependent and independent variables. It helps in predicting the continuous variables such as prediction of Market Trends, prediction of House prices, etc.

The task of the Regression algorithm is to find the mapping function to map the input variable(x) to the continuous output variable(y).

Example: Suppose we want to do weather forecasting, so for this, we will use the Regression algorithm. In weather prediction, the model is trained on the past data, and once the training is completed, it can easily predict the weather for future days.

Types of Regression Algorithm:

Simple Linear Regression

Multiple Linear Regression

Polynomial Regression

Support Vector Regression

Decision Tree Regression

Random Forest Regression

Difference between Regression and Classification

Decision Tree Classification Algorithm

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset.

It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.

A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

Below diagram explains the general structure of a decision tree:

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.

Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.

The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

? Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

? Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

? Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

? Branch/Sub Tree: A tree formed by splitting the tree.

? Pruning: Pruning is the process of removing the unwanted branches from the tree.

? Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:

Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as Attribute selection measure or ASM. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

Information Gain

Gini Index

1. Information Gain:

Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.

It calculates how much information a feature provides us about a class.

According to the value of information gain, we split the node and build the decision tree.

A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

S= Total number of samples

P(yes)= probability of yes

P(no)= probability of no

2. Gini Index:

Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.

An attribute with the low Gini index should be preferred as compared to the high Gini index.

It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.

Gini index can be calculated using the below formula:

Gini Index= 1- ?jPj2

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree pruning technology used:

Cost Complexity Pruning

Reduced Error Pruning.

Advantages of the Decision Tree

It is simple to understand as it follows the same process which a human follow while making any decision in real-life.

It can be very useful for solving decision-related problems.

It helps to think about all the possible outcomes for a problem.

There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

The decision tree contains lots of layers, which makes it complex.

It may have an overfitting issue, which can be resolved using the Random Forest algorithm.

For more class labels, the computational complexity of the decision tree may increase.

Python Implementation of Decision Tree

Now we will implement the Decision tree using Python. For this, we will use the dataset "user_data.csv," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as LogisticRegression, etc.

Steps will also remain the same, which are given below:

Data Pre-processing step

Fitting a Decision-Tree algorithm to the Training set

Predicting the test result

Test accuracy of the result(Creation of Confusion matrix)

Visualizing the test set result.

1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data_set= pd.read_csv('user_data.csv')
```

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
# Splitting the dataset into training 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
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:

2. Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set. For this, we will import the DecisionTreeClassifier class from sklearn.tree library. Below is the code for it:

```
#Fitting Decision Tree classifier to the training set
From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

In the above code, we have created a classifier object, in which we have passed two main parameters;

"criterion='entropy': Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.

random_state=0": For generating the random states.

Below is the output for this:

```
Out[8]:
```

DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,

max_features=None, max_leaf_nodes=None,

min_impurity_decrease=0.0, min_impurity_split=None,

min_samples_leaf=1, min_samples_split=2,

min_weight_fraction_leaf=0.0, presort=False,

random_state=0, splitter='best')

3. Predicting the test result

Now we will predict the test set result. We will create a new prediction vector y_pred. Below is the code for it:

#Predicting the test set result

y_pred= classifier.predict(x_test)

Output:

In the below output image, the predicted output and real test output are given. We can clearly see that there are some values in the prediction vector, which are different from the real vector values. These are prediction errors.

4. Test accuracy of the result (Creation of Confusion matrix)

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

#Creating the Confusion matrix

from sklearn.metrics import confusion_matrix

cm= confusion_matrix(y_test, y_pred)

Output:

In the above output image, we can see the confusion matrix, which has 6+3= 9 incorrect predictions and62+29=91 correct predictions. Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.

5. Visualizing the training set result:

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the decision tree classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in Below is the code for it:

```
#Visulaizing the trianing set result
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  =0.0
1),
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())
fori, 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('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.

As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

6. Visualizing the test set result:

Visualization of test set result will be similar to the visualization of the training set except that the training set will be replaced with the test set.

```
#Visulaizing the test set result
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  =0.0
1),
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())
fori, 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('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of K number of neighbors

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

Firstly, we will choose the number of neighbors, so we will choose the k=5.

Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:

By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.

A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.

Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

It is simple to implement.

It is robust to the noisy training data

It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

Always needs to determine the value of K which may be complex some time.

The computation cost is high because of calculating the distance between the data points for all the training samples.

Python implementation of the KNN algorithm

To do the Python implementation of the K-NN algorithm, we will use the same problem and dataset which we have used in Logistic Regression. But here we will improve the performance of the model. Below is the problem description:

Problem for K-NN Algorithm: There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network. The dataset contains lots of information but the Estimated Salary and Age we will consider for the independent variable and the Purchased variable is for the dependent variable. Below is the dataset:

Steps to implement the K-NN algorithm:

Data Pre-processing step

Fitting the K-NN algorithm 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:

The Data Pre-processing step will remain exactly the same as Logistic Regression. Below is the code for it:

```
# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
```

```
x= data_set.iloc[:, [2,3]].values

y= data_set.iloc[:, 4].values

# Splitting the dataset into training 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

st_x= StandardScaler()

x_train= st_x.fit_transform(x_train)

x_test= st_x.transform(x_test)
```

By executing the above code, our dataset is imported to our program and well pre-processed. After feature scaling our test dataset will look like:

From the above output image, we can see that our data is successfully scaled.

Fitting K-NN classifier to the Training data:

Now we will fit the K-NN classifier to the training data. To do this we will import the KNeighborsClassifier class of Sklearn Neighbors library. After importing the class, we will create the Classifier object of the class. The Parameter of this class will be

n_neighbors: To define the required neighbors of the algorithm. Usually, it takes 5.

metric='minkowski': This is the default parameter and it decides the distance between the points.

p=2: It is equivalent to the standard Euclidean metric.

And then we will fit the classifier to the training data. Below is the code for it:

```
#Fitting K-NN classifier to the training set

from sklearn.neighbors import KNeighborsClassifier

classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )

classifier.fit(x_train, y_train)
```

Output: By executing the above code, we will get the output as:

```
Out[10]:
```

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',

metric_params=None, n_jobs=None, n_neighbors=5, p=2,

weights='uniform')

Predicting the Test Result: To predict the test set result, we will create a y_pred vector as we did in Logistic Regression. Below is the code for it:

#Predicting the test set result

y_pred= classifier.predict(x_test)

Output:

The output for the above code will be:

Creating the Confusion Matrix:

Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the code for it:

#Creating the Confusion matrix

from sklearn.metrics import confusion_matrix

cm= confusion_matrix(y_test, y_pred)

In above code, we have imported the confusion_matrix function and called it using the variable cm.

Output: By executing the above code, we will get the matrix as below:

In the above image, we can see there are 64+29= 93 correct predictions and 3+4= 7 incorrect predictions, whereas, in Logistic Regression, there were 11 incorrect predictions. So we can say that the performance of the model is improved by using the K-NN algorithm.

Visualizing the Training set result:

Now, we will visualize the training set result for K-NN model. The code will remain same as we did in Logistic Regression, except the name of the graph. Below is the code for it:

#Visulaizing the trianing set result

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 =0.0

```
1),
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(('red','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(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

By executing the above code, we will get the below graph:

The output graph is different from the graph which we have occurred in Logistic Regression. It can be understood in the below points:

As we can see the graph is showing the red point and green points. The green points are for Purchased(1) and Red Points for not Purchased(0) variable.

The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-NN algorithm, i.e., finding the nearest neighbor.

The graph has classified users in the correct categories as most of the users who didn't buy the SUV are in the red region and users who bought the SUV are in the green region.

The graph is showing good result but still, there are some green points in the red region and red points in the green region. But this is no big issue as by doing this model is prevented from overfitting issues.

Hence our model is well trained.

Visualizing the Test set result:

After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as x_train and y_train will be replaced by x_test and y_test.

Below is the code for it:

```
#Visualizing the test set result
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  =0.0
1),
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(('red','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(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

The above graph is showing the output for the test data set. As we can see in the graph, the predicted output is well good as most of the red points are in the red region and most of the green points are in the green region.

However, there are few green points in the red region and a few red points in the green region. So these are the incorrect observations that we have observed in the confusion matrix(7 Incorrect output).

MCQ Exercise on K-Nearest Neighbor (KNN) Algorithm for Machine Learning

1. What is the primary computational challenge associated with the K-Nearest Neighbor (KNN) algorithm when applied to large datasets?

High variance

High bias

High dimensionality

High computational cost

Hide Answer Workspace

Answer:

D) High computational cost

Explanation:

KNN has a high computational cost during the prediction phase, as it needs to calculate the distance of the test instance from all training instances.

2. How does the choice of the value 'k' in KNN affect the bias-variance tradeoff?

Lower 'k' increases bias and decreases variance

Higher 'k' increases variance and decreases bias

Lower 'k' decreases bias and increases variance

Higher 'k' increases bias and decreases variance

Hide Answer Workspace

Answer:

D) Higher 'k' increases bias and decreases variance

Explanation:

A higher value of 'k' makes the model simpler (higher bias) but less sensitive to noise (lower variance).

3. Which distance metric is most commonly used in the KNN algorithm for continuous data?

Manhattan distance

Minkowski distance

Euclidean distance

Hamming distance

Hide Answer Workspace

Answer:

C) Euclidean distance

Explanation:

Euclidean distance is commonly used in KNN for continuous data as it calculates the straight-line distance between two points in Euclidean space.

4. In the context of KNN, what is the effect of using a small value of 'k' on the algorithm's sensitivity to noise?

Decreases sensitivity to noise

Increases sensitivity to noise

Has no effect on sensitivity to noise

Normalizes sensitivity to noise

Hide Answer Workspace

Answer:

B) Increases sensitivity to noise

Explanation:

A small value of 'k' makes the algorithm more sensitive to noise and outliers in the dataset, as fewer neighbors are considered in the classification decision.

5. Which technique can be used to speed up the nearest neighbor search in the KNN algorithm for high-dimensional data?

Dimensionality reduction

Increasing the value of 'k'

Normalization

Using the Manhattan distance

Hide Answer Workspace

Answer:

A) Dimensionality reduction

Explanation:

Dimensionality reduction techniques, such as PCA, can be used to reduce the number of features, thereby speeding up the nearest neighbor search by lowering the computational cost.

Logistic Regression in Machine Learning

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.

In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.

Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:

Note: Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.

Logistic Function (Sigmoid Function):

The sigmoid function is a mathematical function used to map the predicted values to probabilities.

It maps any real value into another value within a range of 0 and 1.

The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.

In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

The dependent variable must be categorical in nature.

The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

We know the equation of the straight line can be written as:

In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.

Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"

Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Python Implementation of Logistic Regression (Binomial)

To understand the implementation of Logistic Regression in Python, we will use the below example:

Example: There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.

For this problem, we will build a Machine Learning model using the Logistic regression algorithm. The dataset is shown in the below image. In this problem, we will predict the purchased variable (Dependent Variable) by using age and salary (Independent variables).

Steps in Logistic Regression: To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

Data Pre-processing step

Fitting Logistic Regression to the Training set

Predicting the test result

Test accuracy of the result(Creation of Confusion matrix)

Visualizing the test set result.

1. Data Pre-processing step: In this step, we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

```
#Data Pre-procesing Step

# importing libraries

import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
#importing datasets
data_set= pd.read_csv('user_data.csv')
```

By executing the above lines of code, we will get the dataset as the output. Consider the given image:

Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4. The output will be:

Now we will split the dataset into a training set and test set. Below is the code for it:

```
# Splitting the dataset into training 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)
```

The output for this is given below:

For test set:

For training set:

In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it:

```
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
```

```
x_train= st_x.fit_transform(x_train)

x_test= st_x.transform(x_test)
```

The scaled output is given below:

2. Fitting Logistic Regression to the Training set:

We have well prepared our dataset, and now we will train the dataset using the training set. For providing training or fitting the model to the training set, we will import the LogisticRegression class of the sklearn library.

After importing the class, we will create a classifier object and use it to fit the model to the logistic regression. Below is the code for it:

```
#Fitting Logistic Regression to the training set

from sklearn.linear_model import LogisticRegression

classifier= LogisticRegression(random_state=0)

classifier.fit(x_train, y_train)
```

Output: By executing the above code, we will get the below output:

```
Out[5]:

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,

intercept_scaling=1, l1_ratio=None, max_iter=100,

multi_class='warn', n_jobs=None, penalty='l2',

random_state=0, solver='warn', tol=0.0001, verbose=0,

warm_start=False)
```

Hence our model is well fitted to the training set.

3. Predicting the Test Result

Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code for it:

```
#Predicting the test set result

y_pred= classifier.predict(x_test)
```

In the above code, we have created a y_pred vector to predict the test set result.

Output: By executing the above code, a new vector (y_pred) will be created under the variable explorer option. It can be seen as:

The above output image shows the corresponding predicted users who want to purchase or not purchase the car.

4. Test Accuracy of the result

Now we will create the confusion matrix here to check the accuracy of the classification. To create it, we need to import the confusion_matrix function of the sklearn library. After importing the function, we will call it using a new variable cm. The function takes two parameters, mainly y_true(the actual values) and y_pred (the targeted value return by the classifier). Below is the code for it:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix()
```

Output:

By executing the above code, a new confusion matrix will be created. Consider the below image:

We can find the accuracy of the predicted result by interpreting the confusion matrix. By above output, we can interpret that 65+24= 89 (Correct Output) and 8+3= 11(Incorrect Output).

5. Visualizing the training set result

Finally, we will visualize the training set result. To visualize the result, we will use ListedColormap class of matplotlib library. Below is the code for it:

```
#Visualizing the training set result
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  =0.0
1),
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('Logistic Regression (Training set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()
```

In the above code, we have imported the ListedColormap class of Matplotlib library to create the colormap for visualizing the result. We have created two new variables x_set and y_set to replace x_train and y_train. After that, we have used the nm.meshgrid command to create a rectangular grid, which has a range of -1(minimum) to 1 (maximum). The pixel points we have taken are of 0.01 resolution.

To create a filled contour, we have used mtp.contourf command, it will create regions of provided colors (purple and green). In this function, we have passed the classifier.predict to show the predicted data points predicted by the classifier.

Output: By executing the above code, we will get the below output:

The graph can be explained in the below points:

In the above graph, we can see that there are some Green points within the green region and Purple points within the purple region.

All these data points are the observation points from the training set, which shows the result for purchased variables.

This graph is made by using two independent variables i.e., Age on the x-axis and Estimated salary on the y-axis.

The purple point observations are for which purchased (dependent variable) is probably 0, i.e., users who did not purchase the SUV car.

The green point observations are for which purchased (dependent variable) is probably 1 means user who purchased the SUV car.

We can also estimate from the graph that the users who are younger with low salary, did not purchase the car, whereas older users with high estimated salary purchased the car.

But there are some purple points in the green region (Buying the car) and some green points in the purple region(Not buying the car). So we can say that younger users with a high estimated salary purchased the car, whereas an older user with a low estimated salary did not purchase the car.

The goal of the classifier:

We have successfully visualized the training set result for the logistic regression, and our goal for this classification is to divide the users who purchased the SUV car and who did not purchase the car. So from the output graph, we can clearly see the two regions (Purple and Green) with the observation points. The Purple region is for those users who didn't buy the car, and Green Region is for those users who purchased the car.

Linear Classifier:

As we can see from the graph, the classifier is a Straight line or linear in nature as we have used the Linear model for Logistic Regression. In further topics, we will learn for non-linear Classifiers.

Visualizing the test set result:

Our model is well trained using the training dataset. Now, we will visualize the result for new observations (Test set). The code for the test set will remain same as above except that here we will use x_test and y_test instead of x_train and y_train. Below is the code for it:

#Visulaizing the test set result

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 =0.0 1),

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('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

The above graph shows the test set result. As we can see, the graph is divided into two regions (Purple and Green). And Green observations are in the green region, and Purple observations are in the purple region. So we can say it is a good prediction and model. Some of the green and purple data points are in different regions, which can be ignored as we have already calculated this error using the confusion matrix (11 Incorrect output).

Hence our model is pretty good and ready to make new predictions for this classification problem.

Support Vector Machine Algorithm

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:

SVM algorithm can be used for Face detection, image classification, text categorization, etc.

Types of SVM

SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

How does SVM works?

Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:

So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.

Non-Linear SVM:

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:

So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

z=x2 +y2

By adding the third dimension, the sample space will become as below image:

So now, SVM will divide the datasets into classes in the following way. Consider the below image:

Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:

Hence we get a circumference of radius 1 in case of non-linear data.

Python Implementation of Support Vector Machine

Now we will implement the SVM algorithm using Python. Here we will use the same dataset user_data, which we have used in Logistic regression and KNN classification.

Data Pre-processing step

Till the Data pre-processing step, the code will remain the same. Below is the code:

```
#Data Pre-processing Step

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable

x= data_set.iloc[:, [2,3]].values

y= data_set.iloc[:, 4].values

# Splitting the dataset into training 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

st_x= StandardScaler()

x_train= st_x.fit_transform(x_train)

x_test= st_x.transform(x_test)
```

After executing the above code, we will pre-process the data. The code will give the dataset as:

The scaled output for the test set will be:

Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import SVC class from Sklearn.svm library. Below is the code for it:

```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

In the above code, we have used kernel='linear', as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(x_train, y_train)

Output:

```
Out[8]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
kernel='linear', max_iter=-1, probability=False, random_state=0,
shrinking=True, tol=0.001, verbose=False)
```

The model performance can be altered by changing the value of C(Regularization factor), gamma, and kernel.

Predicting the test set result:

Now, we will predict the output for test set. For this, we will create a new vector y_pred. Below is the code for it:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

After getting the y_pred vector, we can compare the result of y_pred and y_test to check the difference between the actual value and predicted value.

Output: Below is the output for the prediction of the test set:

Creating the confusion matrix:

Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the confusion_matrix function of the sklearn library. After importing the function, we will call it using a new variable cm. The function takes two parameters, mainly y_true(the actual values) and y_pred (the targeted value return by the classifier). Below is the code for it:

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

Output:

As we can see in the above output image, there are 66+24= 90 correct predictions and 8+2= 10 correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

Visualizing the training set result:

Now we will visualize the training set result, below is the code for it:

```
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  =0.0
1),
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(('red', '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(('red', 'green'))(i), label = j)
```

mtp.title('SVM classifier (Training set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

Output:

By executing the above code, we will get the output as:

As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have used a linear kernel in the classifier. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

Visualizing the test set result:

```
#Visulaizing the test set result

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  =0.0
1),

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(('red','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(('red', 'green'))(i), label = j)

mtp.title('SVM classifier (Test set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')
```

mtp.legend()

mtp.show()

Output:

By executing the above code, we will get the output as:

As we can see in the above output image, the SVM classifier has divided the users into two regions (Purchased or Not purchased). Users who purchased the SUV are in the red region with the red scatter points. And users who did not purchase the SUV are in the green region with green scatter points. The hyperplane has divided the two classes into Purchased and not purchased variable.

Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:

Note: To better understand the Random Forest Algorithm, you should have knowledge of the Decision Tree Algorithm.

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.

The predictions from each tree must have very low correlations.

Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

<="" li="" style="box-sizing: border-box;">

It takes less training time as compared to other algorithms.

It predicts output with high accuracy, even for the large dataset it runs efficiently.

It can also maintain accuracy when a large proportion of data is missing.

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

The working of the algorithm can be better understood by the below example:

Example: Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:

Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

Banking: Banking sector mostly uses this algorithm for the identification of loan risk.

Medicine: With the help of this algorithm, disease trends and risks of the disease can be identified.

Land Use: We can identify the areas of similar land use by this algorithm.

Marketing: Marketing trends can be identified using this algorithm.

Advantages of Random Forest

Random Forest is capable of performing both Classification and Regression tasks.

It is capable of handling large datasets with high dimensionality.

It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Python Implementation of Random Forest Algorithm

Now we will implement the Random Forest Algorithm tree using Python. For this, we will use the same dataset "user_data.csv", which we have used in previous classification models. By using the same dataset, we can compare the Random Forest classifier with other classification models such as   etc.

Implementation Steps are given below:

Data Pre-processing step

Fitting the Random forest algorithm to the Training set

Predicting the test result

Test accuracy of the result (Creation of Confusion matrix)

Visualizing the test set result.

1.Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets

data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable

x= data_set.iloc[:, [2,3]].values

y= data_set.iloc[:, 4].values

# Splitting the dataset into training 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

st_x= StandardScaler()

x_train= st_x.fit_transform(x_train)

x_test= st_x.transform(x_test)
```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:

2. Fitting the Random Forest algorithm to the training set:

Now we will fit the Random forest algorithm to the training set. To fit it, we will import the RandomForestClassifier class from the sklearn.ensemble library. The code is given below:

```
#Fitting Decision Tree classifier to the training set

from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")

classifier.fit(x_train, y_train)
```

In the above code, the classifier object takes below parameters:

n_estimators= The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.

criterion= It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

Output:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=10,
n_jobs=None, oob_score=False, random_state=None,
verbose=0, warm_start=False)
```

3. Predicting the Test Set result

Since our model is fitted to the training set, so now we can predict the test result. For prediction, we will create a new prediction vector y_pred. Below is the code for it:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

Output:

The prediction vector is given as:

By checking the above prediction vector and test set real vector, we can determine the incorrect predictions done by the classifier.

4. Creating the Confusion Matrix

Now we will create the confusion matrix to determine the correct and incorrect predictions. Below is the code for it:

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

Output:

As we can see in the above matrix, there are 4+4= 8 incorrect predictions and 64+28= 92 correct predictions.

5. Visualizing the training Set result

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the Random forest classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in Below is the code for it:

```
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  =0.0
1),
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('Random Forest Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

The above image is the visualization result for the Random Forest classifier working with the training set result. It is very much similar to the Decision tree classifier. Each data point corresponds to each user of the user_data, and the purple and green regions are the prediction regions. The purple region is classified for the users who did not purchase the SUV car, and the green region is for the users who purchased the SUV.

So, in the Random Forest classifier, we have taken 10 trees that have predicted Yes or NO for the Purchased variable. The classifier took the majority of the predictions and provided the result.

6. Visualizing the test set result

Now we will visualize the test set result. Below is the code for it:

```
#Visulaizing the test set result
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  =0.0
1),
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('Random Forest Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

The above image is the visualization result for the test set. We can check that there is a minimum number of incorrect predictions (8) without the Overfitting issue. We will get different results by changing the number of trees in the classifier.

Machine Learning Interview Questions and Answers

A list of frequently asked machine learning interview questions and answers are given below.

1) What do you understand by Machine learning?

Machine learning is the form of Artificial Intelligence that deals with system programming and automates data analysis to enable computers to learn and act through experiences without being explicitly programmed.

For example, Robots are coded in such a way that they can perform the tasks based on data they collect from sensors. They automatically learn programs from data and improve with experiences.

2) Differentiate between inductive learning and deductive learning?

In inductive learning, the model learns by examples from a set of observed instances to draw a generalized conclusion. On the other side, in deductive learning, the model first applies the conclusion, and then the conclusion is drawn.

Inductive learning is the method of using observations to draw conclusions.

Deductive learning is the method of using conclusions to form observations.

For example, if we have to explain to a kid that playing with fire can cause burns. There are two ways we can explain this to a kid; we can show training examples of various fire accidents or images of burnt people and label them as "Hazardous". In this case, a kid will understand with the help of examples and not play with the fire. It is the form of Inductive machine learning. The other way to teach the same thing is to let the kid play with the fire and wait to see what happens. If the kid gets a burn, it will teach the kid not to play with fire and avoid going near it. It is the form of deductive learning.

3) What is the difference between Data Mining and Machine Learning?

Data mining can be described as the process in which the structured data tries to abstract knowledge or interesting unknown patterns. During this process, machine learning algorithms are used.

Machine learning represents the study, design, and development of the algorithms which provide the ability to the processors to learn without being explicitly programmed.

4) What is the meaning of Overfitting in Machine learning?

Overfitting can be seen in machine learning when a statistical model describes random error or noise instead of the underlying relationship. Overfitting is usually observed when a model is excessively complex. It happens because of having too many parameters concerning the number of training data types. The model displays poor performance, which has been overfitted.

5) Why overfitting occurs?

The possibility of overfitting occurs when the criteria used for training the model is not as per the criteria used to judge the efficiency of a model.

6) What is the method to avoid overfitting?

Overfitting occurs when we have a small dataset, and a model is trying to learn from it. By using a large amount of data, overfitting can be avoided. But if we have a small database and are forced to build a model based on that, then we can use a technique known as cross-validation. In this method, a model is usually given a dataset of a known data on which training data set is run and dataset of unknown data against which the model is tested. The primary aim of cross-validation is to define a dataset to "test" the model in the training phase. If there is sufficient data, 'Isotonic Regression' is used to prevent overfitting.

7) Differentiate supervised and unsupervised machine learning.

In supervised machine learning, the machine is trained using labeled data. Then a new dataset is given into the learning model so that the algorithm provides a positive outcome by analyzing the labeled data. For example, we first require to label the data which is necessary to train the model while performing classification.

In the unsupervised machine learning, the machine is not trained using labeled data and let the algorithms make the decisions without any corresponding output variables.

8) How does Machine Learning differ from Deep Learning?

Machine learning is all about algorithms which are used to parse data, learn from that data, and then apply whatever they have learned to make informed decisions.

Deep learning is a part of machine learning, which is inspired by the structure of the human brain and is particularly useful in feature detection.

9) How is KNN different from k-means?

KNN or K nearest neighbors is a supervised algorithm which is used for classification purpose. In KNN, a test sample is given as the class of the majority of its nearest neighbors. On the other side, K-means is an unsupervised algorithm which is mainly used for clustering. In k-means clustering, it needs a set of unlabeled points and a threshold only. The algorithm further takes unlabeled data and learns how to cluster it into groups by computing the mean of the distance between different unlabeled points.

10) What are the different types of Algorithm methods in Machine Learning?

The different types of algorithm methods in machine earning are:

Supervised Learning

Semi-supervised Learning

Unsupervised Learning

Transduction

Reinforcement Learning

11) What do you understand by Reinforcement Learning technique?

Reinforcement learning is an algorithm technique used in Machine Learning. It involves an agent that interacts with its environment by producing actions & discovering errors or rewards. Reinforcement learning is employed by different software and machines to search for the best suitable behavior or path it should follow in a specific situation. It usually learns on the basis of reward or penalty given for every action it performs.

12) What is the trade-off between bias and variance?

Both bias and variance are errors. Bias is an error due to erroneous or overly simplistic assumptions in the learning algorithm. It can lead to the model under-fitting the data, making it hard to have high predictive accuracy and generalize the knowledge from the training set to the test set.

Variance is an error due to too much complexity in the learning algorithm. It leads to the algorithm being highly sensitive to high degrees of variation in the training data, which can lead the model to overfit the data.

To optimally reduce the number of errors, we will need to tradeoff bias and variance.

13) How do classification and regression differ?

14) What are the five popular algorithms we use in Machine Learning?

Five popular algorithms are:

Decision Trees

Probabilistic Networks

Neural Networks

Support Vector Machines

Nearest Neighbor

15) What do you mean by ensemble learning?

Numerous models, such as classifiers are strategically made and combined to solve a specific computational program which is known as ensemble learning. The ensemble methods are also known as committee-based learning or learning multiple classifier systems. It trains various hypotheses to fix the same issue. One of the most suitable examples of ensemble modeling is the random forest trees where several decision trees are used to predict outcomes. It is used to improve the classification, function approximation, prediction, etc. of a model.

16) What is a model selection in Machine Learning?

The process of choosing models among diverse mathematical models, which are used to define the same data is known as Model Selection. Model learning is applied to the fields of statistics, data mining, and machine learning.

17) What are the three stages of building the hypotheses or model in machine learning?

There are three stages to build hypotheses or model in machine learning:

Model building

It chooses a suitable algorithm for the model and trains it according to the requirement of the problem.

Applying the model

It is responsible for checking the accuracy of the model through the test data.

Model testing

It performs the required changes after testing and apply the final model.

18) What according to you, is the standard approach to supervised learning?

In supervised learning, the standard approach is to split the set of example into the training set and the test.

19) Describe 'Training set' and 'training Test'.

In various areas of information of machine learning, a set of data is used to discover the potentially predictive relationship, which is known as 'Training Set'. The training set is an example that is given to the learner. Besides, the 'Test set' is used to test the accuracy of the hypotheses generated by the learner. It is the set of instances held back from the learner. Thus, the training set is distinct from the test set.

20) What are the common ways to handle missing data in a dataset?

Missing data is one of the standard factors while working with data and handling. It is considered as one of the greatest challenges faced by the data analysts. There are many ways one can impute the missing values. Some of the common methods to handle missing data in datasets can be defined as deleting the rows, replacing with mean/median/mode, predicting the missing values, assigning a unique category, using algorithms that support missing values, etc.

21) What do you understand by ILP?

ILP stands for Inductive Logic Programming. It is a part of machine learning which uses logic programming. It aims at searching patterns in data which can be used to build predictive models. In this process, the logic programs are assumed as a hypothesis.

22) What are the necessary steps involved in Machine Learning Project?

There are several essential steps we must follow to achieve a good working model while doing a Machine Learning Project. Those steps may include parameter tuning, data preparation, data collection, training the model, model evaluation, and prediction, etc.

23) Describe Precision and Recall?

Precision and Recall both are the measures which are used in the information retrieval domain to measure how good an information retrieval system reclaims the related data as requested by the user.

Precision can be said as a positive predictive value. It is the fraction of relevant instances among the received instances.

On the other side, recall is the fraction of relevant instances that have been retrieved over the total amount or relevant instances. The recall is also known as sensitivity.

24) What do you understand by Decision Tree in Machine Learning?

Decision Trees can be defined as the Supervised Machine Learning, where the data is continuously split according to a certain parameter. It builds classification or regression models as similar as a tree structure, with datasets broken up into ever smaller subsets while developing the decision tree. The tree can be defined by two entities, namely decision nodes, and leaves. The leaves are the decisions or the outcomes, and the decision nodes are where the data is split. Decision trees can manage both categorical and numerical data.

25) What are the functions of Supervised Learning?

Classification

Speech Recognition

Regression

Predict Time Series

Annotate Strings

26) What are the functions of Unsupervised Learning?

Finding clusters of the data

Finding low-dimensional representations of the data

Finding interesting directions in data

Finding novel observations/ database cleaning

Finding interesting coordinates and correlations

27) What do you understand by algorithm independent machine learning?

Algorithm independent machine learning can be defined as machine learning, where mathematical foundations are independent of any particular classifier or learning algorithm.

28) Describe the classifier in machine learning.

A classifier is a case of a hypothesis or discrete-valued function which is used to assign class labels to particular data points. It is a system that inputs a vector of discrete or continuous feature values and outputs a single discrete value, the class.

29) What do you mean by Genetic Programming?

Genetic Programming (GP) is almost similar to an Evolutionary Algorithm, a subset of machine learning. Genetic programming software systems implement an algorithm that uses random mutation, a fitness function, crossover, and multiple generations of evolution to resolve a user-defined task. The genetic programming model is based on testing and choosing the best option among a set of results.

30) What is SVM in machine learning? What are the classification methods that SVM can handle?

SVM stands for Support Vector Machine. SVM are supervised learning models with an associated learning algorithm which analyze the data used for classification and regression analysis.

The classification methods that SVM can handle are:

Combining binary classifiers

Modifying binary to incorporate multiclass learning

31) How will you explain a linked list and an array?

An array is a datatype which is widely implemented as a default type, in almost all the modern programming languages. It is used to store data of a similar type.

But there are many use-cases where we don't know the quantity of data to be stored. For such cases, advanced data structures are required, and one such data structure is linked list.

There are some points which explain how the linked list is different from an array:

32) What do you understand by the Confusion Matrix?

A confusion matrix is a table which is used for summarizing the performance of a classification algorithm. It is also known as the error matrix.

Where,

TN= True Negative

TP= True Positive

FN= False Negative

FP= False Positive

33) Explain True Positive, True Negative, False Positive, and False Negative in Confusion Matrix with an example.

True Positive

When a model correctly predicts the positive class, it is said to be a true positive.

For example, Umpire gives a Batsman NOT OUT when he is NOT OUT.

True Negative

When a model correctly predicts the negative class, it is said to be a true negative.

For example, Umpire gives a Batsman OUT when he is OUT.

False Positive

When a model incorrectly predicts the positive class, it is said to be a false positive. It is also known as 'Type I' error.

For example, Umpire gives a Batsman NOT OUT when he is OUT.

False Negative

When a model incorrectly predicts the negative class, it is said to be a false negative. It is also known as 'Type II' error.

For example, Umpire gives a Batsman OUT when he is NOT OUT.

34) What according to you, is more important between model accuracy and model performance?

Model accuracy is a subset of model performance. The accuracy of the model is directly proportional to the performance of the model. Thus, better the performance of the model, more accurate are the predictions.

35) What is Bagging and Boosting?

Bagging is a process in ensemble learning which is used for improving unstable estimation or classification schemes.

Boosting methods are used sequentially to reduce the bias of the combined model.

36) What are the similarities and differences between bagging and boosting in Machine Learning?

Similarities of Bagging and Boosting

Both are the ensemble methods to get N learns from 1 learner.

Both generate several training data sets with random sampling.

Both generate the final result by taking the average of N learners.

Both reduce variance and provide higher scalability.

Differences between Bagging and Boosting

Although they are built independently, but for Bagging, Boosting tries to add new models which perform well where previous models fail.

Only Boosting determines the weight for the data to tip the scales in favor of the most challenging cases.

Only Boosting tries to reduce bias. Instead, Bagging may solve the problem of over-fitting while boosting can increase it.

37) What do you understand by Cluster Sampling?

Cluster Sampling is a process of randomly selecting intact groups within a defined population, sharing similar characteristics. Cluster sample is a probability where each sampling unit is a collection or cluster of elements.

For example, if we are clustering the total number of managers in a set of companies, in that case, managers (sample) will represent elements and companies will represent clusters.

38) What do you know about Bayesian Networks?

Bayesian Networks also referred to as 'belief networks' or 'casual networks', are used to represent the graphical model for probability relationship among a set of variables.

For example, a Bayesian network can be used to represent the probabilistic relationships between diseases and symptoms. As per the symptoms, the network can also compute the probabilities of the presence of various diseases.

Efficient algorithms can perform inference or learning in Bayesian networks. Bayesian networks which relate the variables (e.g., speech signals or protein sequences) are called dynamic Bayesian networks.

39) Which are the two components of Bayesian logic program?

A Bayesian logic program consists of two components:

Logical

It contains a set of Bayesian Clauses, which capture the qualitative structure of the domain.

Quantitative

It is used to encode quantitative information about the domain.

40) Describe dimension reduction in machine learning.

Dimension reduction is the process which is used to reduce the number of random variables under considerations.

Dimension reduction can be divided into feature selection and extraction.

41) Why instance-based learning algorithm sometimes referred to as Lazy learning algorithm?

In machine learning, lazy learning can be described as a method where induction and generalization processes are delayed until classification is performed. Because of the same property, an instance-based learning algorithm is sometimes called lazy learning algorithm.

42) What do you understand by the F1 score?

The F1 score represents the measurement of a model's performance. It is referred to as a weighted average of the precision and recall of a model. The results tending to 1 are considered as the best, and those tending to 0 are the worst. It could be used in classification tests, where true negatives don't matter much.

43) How is a decision tree pruned?

Pruning is said to occur in decision trees when the branches which may consist of weak predictive power are removed to reduce the complexity of the model and increase the predictive accuracy of a decision tree model. Pruning can occur bottom-up and top-down, with approaches such as reduced error pruning and cost complexity pruning.

Reduced error pruning is the simplest version, and it replaces each node. If it is unable to decrease predictive accuracy, one should keep it pruned. But, it usually comes pretty close to an approach that would optimize for maximum accuracy.

44) What are the Recommended Systems?

Recommended System is a sub-directory of information filtering systems. It predicts the preferences or rankings offered by a user to a product. According to the preferences, it provides similar recommendations to a user. Recommendation systems are widely used in movies, news, research articles, products, social tips, music, etc.

45) What do you understand by Underfitting?

Underfitting is an issue when we have a low error in both the training set and the testing set. Few algorithms work better for interpretations but fail for better predictions.

46) When does regularization become necessary in Machine Learning?

Regularization is necessary whenever the model begins to overfit/ underfit. It is a cost term for bringing in more features with the objective function. Hence, it tries to push the coefficients for many variables to zero and reduce cost term. It helps to reduce model complexity so that the model can become better at predicting (generalizing).

47) What is Regularization? What kind of problems does regularization solve?

A regularization is a form of regression, which constrains/ regularizes or shrinks the coefficient estimates towards zero. In other words, it discourages learning a more complex or flexible model to avoid the risk of overfitting. It reduces the variance of the model, without a substantial increase in its bias.

Regularization is used to address overfitting problems as it penalizes the loss function by adding a multiple of an L1 (LASSO) or an L2 (Ridge) norm of weights vector w.

48) Why do we need to convert categorical variables into factor? Which functions are used to perform the conversion?

Most Machine learning algorithms require number as input. That is why we convert categorical values into factors to get numerical values. We also don't have to deal with dummy variables.

The functions factor() and as.factor() are used to convert variables into factors.

49) Do you think that treating a categorical variable as a continuous variable would result in a better predictive model?

For a better predictive model, the categorical variable can be considered as a continuous variable only when the variable is ordinal in nature.

50) How is machine learning used in day-to-day life?

Most of the people are already using machine learning in their everyday life. Assume that you are engaging with the internet, you are actually expressing your preferences, likes, dislikes through your searches. All these things are picked up by cookies coming on your computer, from this, the behavior of a user is evaluated. It helps to increase the progress of a user through the internet and provide similar suggestions.

The navigation system can also be considered as one of the examples where we are using machine learning to calculate a distance between two places using optimization techniques. Surely, people are going to more engage with machine learning in the near future.