LOURDES MATHA COLLEGE OF SCIENCE AND TECHNOLOGY

KUTTICHAL P O,THIRUVANANTHAPURAM, Kerala -695574

DATA SCIENCE LAB

20MCA241

Name : … JACHITHRA C S…..……

University Reg No: .. LMC20MCA-2014…….

Class: …MCA……Semester……S3…….

Year: ……2020-2022……..…...

From Page No: ……1…To……48....……

Certified Bonafide Record of Work Done By

….……JACHITHRA C S ……

Place: ..Kuttichal.……	Head of Dept	Professor-in-Charge
Date: …25/03/2022….	SELMA JOSEPH	Anjana J

External Examiner

INDEX

Data Science Introduction

Data Science is about finding patterns in data, through analysis, and make future predictions. By using data science ,companies are able to make Better decisions

Data Science **Life Cycle**

Step 1: Define Problem Statement

Creating a well-defined problem statement is a first and critical step in data science. It is a brief description of the problem that you are going to solve.

Step 2: Data Collection

You need to collect the data which can help to solve the problem. Data collection is a systematic approach to gather relevant information from a variety of sources. Depending on the problem statement, the data collection method is broadly classified into two categories.

First, when you have some unique problem and no related research is done on the subject. Then, you need to collect new data. This method is called as primary data collection. For example, you want information on the average time that employees spend in a cafeteria across companies. There is no public data available of these. But you can collect the data through various methods such as surveys, interviews of employees and by monitoring the time spent by employees in cafeteria. This method is time-consuming.

Another method is to use the data which is readily available or collected by someone else. These data can be found on the internet, news articles, government census, magazines and so on. This method is called as secondary data collection. This method is less time-consuming than the primary method.

For our problem statement on EPL. You can collect and aggregate the data from various open-source websites such as Github, Kaggle, and datahub.

Name	Age	OVA	POT	Team& Contract	ID	Height	Weight	foot	BOV	BP
P. Aubameyang ST LM	29	88	88	Arsenal 2018 ~ 2021	188567	6'2"	176lbs	Right	88	ST
A. Lacazette ST	27	85	85	Arsenal 2017 ~ 2022	193301	5'9"	161lbs	Right	86	CF
D. Welbeck ST LW	27	79	79	Arsenal 2014 ~ 2019	186146	6'1"	161lbs	Right	80	ST
E. Nketiah ST	19	66	83	Arsenal 2015 ~ 2021	236988	5'9"	154lbs	Right	67	CF
T. Abraham ST On Loan	20	75	85	Aston Villa Jun 30, 2019	231352	6'3"	176lbs	Right	76	ST
J. Kodjia ST LW	28	73	73	Aston Villa 2016 ~ 2020	189099	6'2"	170lbs	Right	74	ST
B. Bjarnason CM CDM	30	69	69	Aston Villa 2017 ~ 2020	175789	6'0"	170lbs	Right	72	ST
K. Davis ST	20	65	76	Aston Villa 2015 ~ 2020	231743	6'2"	165lbs	Right	65	ST
C. Wilson ST	26	80	81	Bournemouth 2014 ~ 2022	196978	5'11"	146lbs	Right	81	ST
J. King CF ST	26	79	80	Bournemouth 2015 ~ 2021	185422	6'0"	181lbs	Right	81	CF
L. Mousset ST	22	70	75	Bournemouth 2016 ~ 2020	221618	6'0"	176lbs	Right	71	ST
D. Solanke ST	20	70	83	Bournemouth 2019 ~ 2023	225539	6'2"	176lbs	Right	71	ST
S. Surridge ST	19	65	77	Bournemouth 2016 ~ 2021	234249	6'3"	170lbs	Right	66	ST
J. Anthony ST	18	59	73	Bournemouth 2018 ~ 2019	243669	6'0"	150lbs	Right	59	ST
G. Murray ST	34	77	77	Brighton & Hove Albion 2017 ~ 2020	172937	6'0"	176lbs	Right	77	ST
F. Andone ST	25	77	79	Brighton & Hove Albion 2018 ~ 2023	225018	5'11"	172lbs	Right	78	ST
J. Locadia ST LW	24	76	79	Brighton & Hove Albion 2018 ~ 2022	204366	6'1"	192lbs	Right	76	ST
V. GyÃ¶keres ST LW RW	20	62	76	Brighton & Hove Albion 2018 ~ 2022	241651	6'2"	190lbs	Right	63	ST

Table 1: Snapshot of data collected from web sources

Step 3: Data Quality Check and Remediation

One of the most important and often ignored aspects by data scientists is ensuring the data that is used for analysis and interpretation is of good quality.

After collecting the data, most people start the analysis on it. Often, they forgot to do a sanity check on the data. If the data is of bad quality, it can give misleading information. Simply said: "Garbage in, garbage Out".

For example, if you start the analysis without ensuring data quality. Then you might get unexpected results such as the Crystal Palace club will win the next EPL. However, your domain knowledge on EPL says that the result looks inaccurate as Crystal Palace has never even finished in the top 4.

Step 4: Exploratory Data Analysis

Before you model the steps to arrive at a solution, it's important to analyse the data. It is the most exciting step as it helps you to build familiarity with the data and extract useful insights. If you skip this step then you might end up generating inaccurate models and choosing the insignificant variables in your model.

As quoted by John Tukey, developer of Exploratory Data Analysis,

"It is important to understand what you CAN DO before you learn to measure how WELL you seem to have DONE it."

But it can be strenuous and difficult. Are there tools or techniques to explore data in a meaningful way?

Yes, you can use descriptive statistics such as central value measures and variability measures. Also, visualisation methods such as graphs and plots can be used for analysis.

For example, you can calculate the average goals scored per match. You can also check if home advantage is a real thing?

The bar chart below shows the home advantage of Liverpool. It shows that of all home matches of Liverpool, they have won 17 and drawn 2. But have never lost any home match.

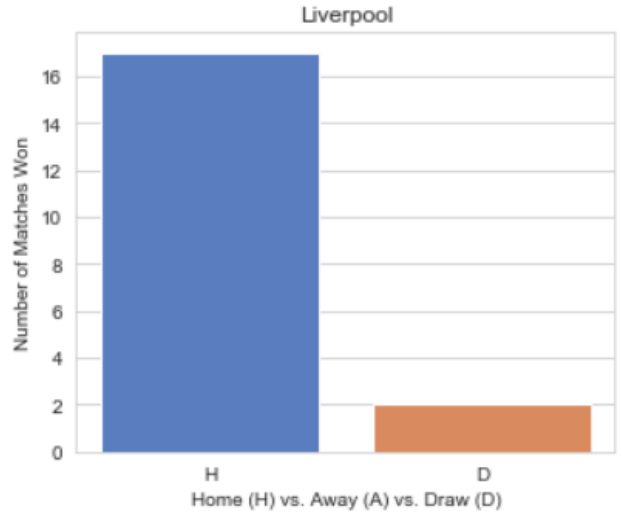


Chart 1: Home Advantage for Liverpool [1]

However, there is no home advantage of Leicester. It shows that of all home matches of Leicester, they have won 8, lost 8 and drawn 3.

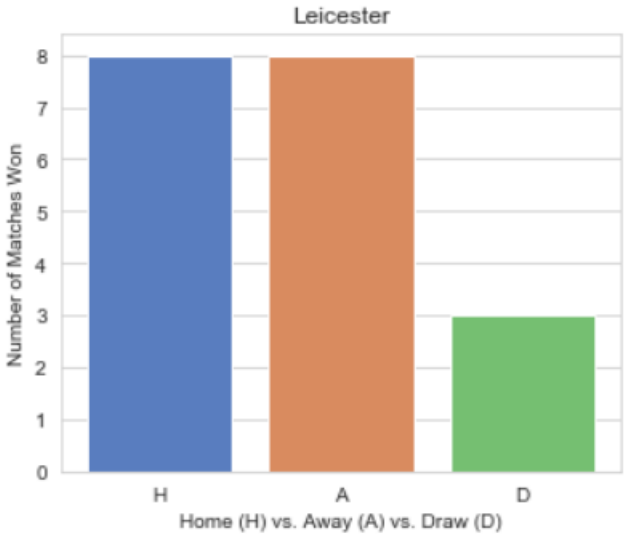


Chart 2: No home Advantage for Leicester [1]

Source: Intro to Data Science by Quantra

Data analysis is an iterative process that helps you to get closer to the solution. Every iteration has a cost associated with it. Therefore, it is advisable that as a data scientist you plan properly so that the number of iterations is reduced. You can play with data, create your own graphs and learn to derive inferences. All these analyses which you performed are commonly known as exploratory data analysis (EDA).

Step 5: Data Modelling

Modelling means formulating every step and gather the techniques required to achieve the solution. You need to list down the flow of the calculations which is nothing but modelling steps to the solution. The

important factor is how to perform the calculations. There are various techniques under Statistics and Machine Learning that you can choose based on the requirement.

For EPL data, we chose to use statistical techniques to predict the winner in the current season.

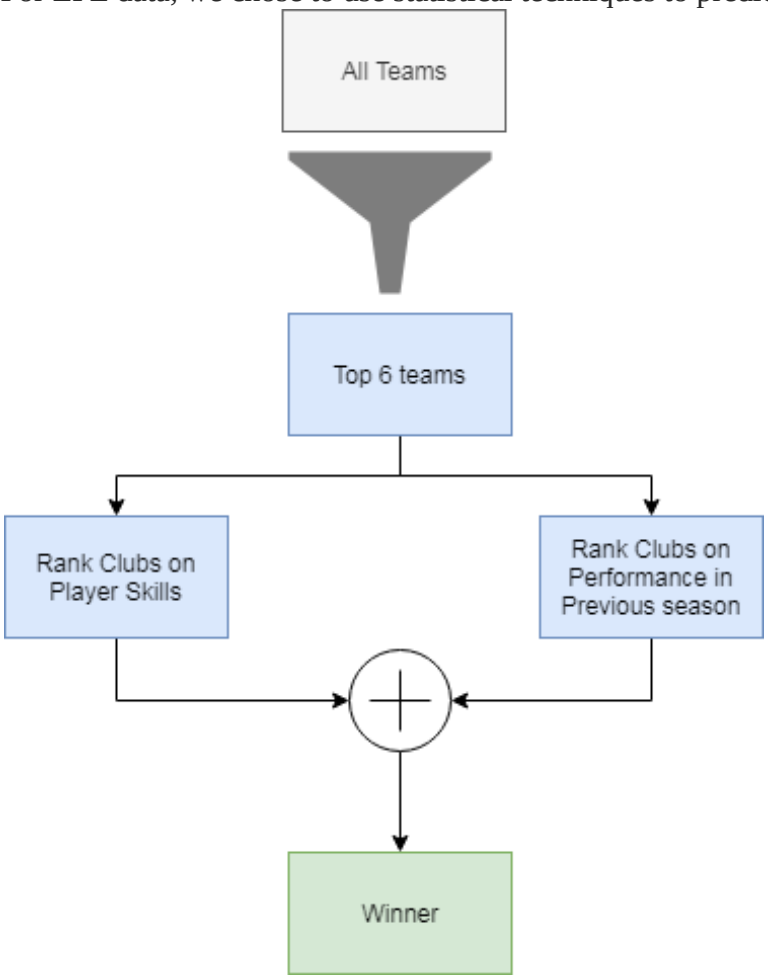


Figure 1: Data Modelling Steps [1]

Top 6 teams

As per the data of the last three seasons, we have learnt that the winner of EPL is changing but the top 6 clubs remain the same. These clubs are Arsenal, Chelsea, Liverpool, Manchester City, Manchester United and Tottenham Hotspur. The winner in the next season is very likely to be from these top 6 teams.

Rank Clubs on Player Skills

As we know, in football a player can be an Attacker, Defender, Midfielder or a Goalkeeper. Each of these positions requires a different set of physical skills. A player is assigned a score based on his physical skills and the position he is playing in. The scores are given on a scale of 0–100. And each of these skills is assigned a weight.

Club	Players Skill-based Rating	Ranking
Tottenham Hotspur	83.06	1
Manchester City	82.93	2
Liverpool	82.29	3
Manchester United	81.53	4
Arsenal	80.01	5
Chelsea	79.95	6

Table 3: Player skills by Club. [1] and [2]

Rank Clubs on Past Performance

We rank the performance of the teams based on the number of wins and difference in goals in the last season.

Club	Goal Difference Ranking	No of Wins Ranking	Previous Performance Ranking
Tottenham Hotspur	3	3	3
Manchester City	1	1	1
Liverpool	2	2	2
Manchester United	6	6	6
Arsenal	5	4	5
Chelsea	4	4	4

Table 4: Club last season performance.

Combine the Ranks to predict the winner

Considering 80% weight to the players' skills and 20% weight to the past performance of clubs we get the final ranking of the top 6 clubs.
Based on these calculations, Tottenham Hotspur is likely to win this EPL.

Club	Players Skill-based Ranking (80%)	Previous Performance Ranking (20%)	Final Ranking
Tottenham Hotspur	1	3	1
Manchester City	2	1	2
Liverpool	3	2	3
Manchester United	4	6	4
Arsenal	5	5	5
Chelsea	6	4	6

Table 5: Combining Player skills and Past Performance

Step 6: Data Communication

Source: Freepik

This is the final step where you present the results from your analysis to the stakeholders. You explain to them how you came to a specific conclusion and your critical findings.

Most often you need to present your findings to a non-technical audience, such as the marketing team or business executives. You need to communicate the results in a simple to understand manner. And the stakeholders should be able to chalk out an actionable plan from it.

But what are the important things you need to keep in mind while communicating your results?

1. **Know your audience and speak their language**

You should know your audience and speak their language. For example, you are presenting the prediction of EPL winner to a football fan. They do not understand the statistics you have used but they can relate to the steps you followed to decide the winner.

2. Focus on values and outcomes

You should focus on value and outcomes. Your stakeholder would not be interested in how you got the data but from where you got the data. The usage of credible sources helps to build confidence in your prediction.

3. Communicate assumptions and limitations

You should clearly communicate the important assumptions and limitations you have made. For example, to calculate the overall rating of the team you have assumed 3–4–3–1 formation for all the teams. It's important that you communicate these to the stakeholders.

Although the goal is to predict the winner, there might be some other critical findings that are relevant. For example, the teams with the best attack, midfield, defence and goalkeeping. The best player for each position in the league based on the players' skills. Such a unified view of the data in the form of graphs, charts and numbers is called a Dashboard. you can use Excel to create the dashboard.

Knn classification

Working

For a given data point in the set, the algorithms find the distances between this and all other **K** numbers of datapoint in the dataset close to the initial point and votes for that category that has the most frequency. Usually, **Euclidean distance** is taking as a measure of distance. Thus the end resultant model is just the labeled data placed in a space. This algorithm is popularly known for various applications like **genetics**, **forecasting**, etc. The algorithm is best when more features are present and out shows SVM in this case.

KNN reducing overfitting is a fact. On the other hand, there is a need to choose the best value for K. So now how do we choose K? Generally we use the Square root of the number of samples in the dataset as value for K. An optimal value has to be found out since lower value may lead to overfitting and higher value may require high computational complication in distance. So using an error plot may help. Another method is the elbow method. You can prefer to take root else can also follow the elbow method.

Let's dive deep into the different steps of K-NN for classifying a new data point

Step 1: Select the value of K neighbors(say k=5)

Step 2: Find the K (5) nearest data point for our new data point based on euclidean distance(which we discuss later)

Step 3: Among these K data points count the data points in each category

Step 4: Assign the new data point to the category that has the most neighbors of the new datapoint

inear Regression vs. Multiple Regression: An Overview

Regression analysis is a common statistical method used in finance and investing. Linear regression is one of the most common techniques of regression analysis. Multiple regression is a broader class of regressions that encompasses linear and nonlinear regressions with multiple explanatory variables.

Regression as a tool helps pool data together to help people and companies make informed decisions. There are different variables at play in regression, including a dependent variable—the main variable that you're trying to understand—and an independent variable—factors that may have an impact on the dependent variable.

In order to make regression analysis work, you must collect all the relevant data. It can be presented on a graph, with an x-axis and a y-axis.

There are several main reasons people use regression analysis:

1. To predict future economic conditions, trends, or values
2. To determine the relationship between two or more variables
3. To understand how one variable changes when another change

There are many different kinds of regression analysis. For the purpose of this article, we will look at two: linear regression and multiple regression.

Linear Regression

It is also called simple linear regression. It establishes the relationship between two variables using a straight line. Linear regression attempts to draw a line that comes closest to the data by finding the slope and intercept that define the line and minimize regression errors.

If two or more explanatory variables have a linear relationship with the dependent variable, the regression is called a multiple linear regression.

Many data relationships do not follow a straight line, so statisticians use nonlinear regression instead. The two are similar in that both track a particular response from a set of variables graphically. But nonlinear models are more complicated than linear models because the function is created through a series of assumptions that may stem from trial and error.

Multiple Regression

It is rare that a dependent variable is explained by only one variable. In this case, an analyst uses multiple regression, which attempts to explain a dependent variable using more than one independent variable. Multiple regressions can be linear and nonlinear.

Multiple regressions are based on the assumption that there is a linear relationship between both the dependent and independent variables. It also assumes no major correlation between the independent variables.

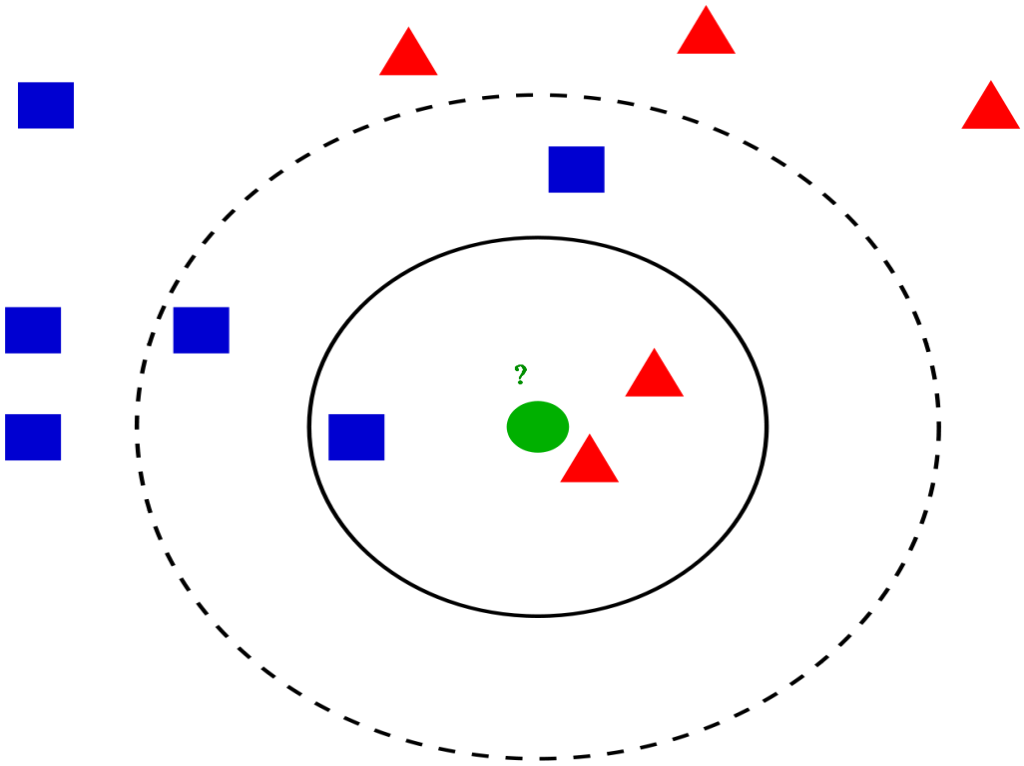
As mentioned above, there are several different advantages to using regression analysis. These models can be used by businesses and economists to help make practical decisions.

A company can not only use regression analysis to understand certain situations like why customer service calls are dropping, but also to make forward-looking predictions like sales figures in the future, and make important decisions like special sales and promotions.

Linear Regression vs. Multiple Regression: Example

Consider an analyst who wishes to establish a linear relationship between the daily change in a company's stock prices and other explanatory variables such as the daily change in trading volume and the daily change in market returns. If he runs a regression with the daily change in the company's stock prices as a dependent variable and the daily change in trading volume as an independent variable, this would be an example of a simple linear regression with one explanatory variable.

If the analyst adds the daily change in market returns into the regression, it would be a multiple linear regression.



Example

Let’s go through an example problem for getting a clear intuition on the K -Nearest Neighbor classification. We are using the Social network ad dataset (Download). The dataset contains the details of users in a social networking site to find whether a user buys a product by clicking the ad on the site based on their salary, age, and gender.

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0

Let's start the programming by importing essential libraries

```
importnumpy as np
importmatplotlib.pyplot as plt
import pandas as pd
importsklearn
```

Importing of the dataset and slicing it into independent and dependent variables

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [1, 2, 3]].values
y = dataset.iloc[:, -1].values
```

Since our dataset containing character variables we have to encode it using LabelEncoder

```
fromsklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
```

We are performing a train test split on the dataset. We are providing the test size as 0.20, that means our training sample contains 320 training set and test sample contains 80 test set

```
fromsklearn.model_selection import train_test_split
```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

Next, we are doing feature scaling to the training and test set of independent variables for reducing the size to smaller values

fromsklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

Now we have to create and train the K Nearest Neighbor model with the training set

fromsklearn.neighbors import KNeighborsClassifier

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

classifier.fit(X_train, y_train)

We are using 3 parameters in the model creation. n_neighbors is setting as 5, which means 5 neighborhood points are required for classifying a given point. The distance metric we are using is Minkowski, the equation for it is given below

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

As per the equation, we have to select the p-value also.

p = 1 , Manhattan Distance

p = 2 , Euclidean Distance

p = infinity.....,.Cheybchev Distance

In our problem, we are choosing the p as 2 (also u can choose the metric as “euclidean”)
Our Model is created, now we have to predict the output for the test set

y_pred = classifier.predict(X_test)

Comparing true and predicted value :

y_test

array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,

0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,

1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,

0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,

1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1], dtype=int64)

y_pred

array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,

0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,

1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,

0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,

1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1], dtype=int64)

We can evaluate our model using the confusion matrix and accuracy score by comparing the predicted and actual test values

fromsklearn.metrics import confusion_matrix,accuracy_score

cm = confusion_matrix(y_test, y_pred)

ac = accuracy_score(y_test,y_pred)

confusion matrix –

[[64 4]

[3 29]]

accuracy is 0.95

Our model is performing well. The full code for the problem is given below.

```
# Importing the libraries
importnumpy as np
importmatplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
```

```
fromsklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Feature Scaling
fromsklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the K-NN model on the Training set
fromsklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)

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

# Making the Confusion Matrix
fromsklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
```

Naive Bayes Classifiers

- Difficulty Level : Medium
- Last Updated : 02 Feb, 2022

This article discusses the theory behind the Naive Bayes classifiers and their implementation.

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.
To start with, let us consider a dataset.

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

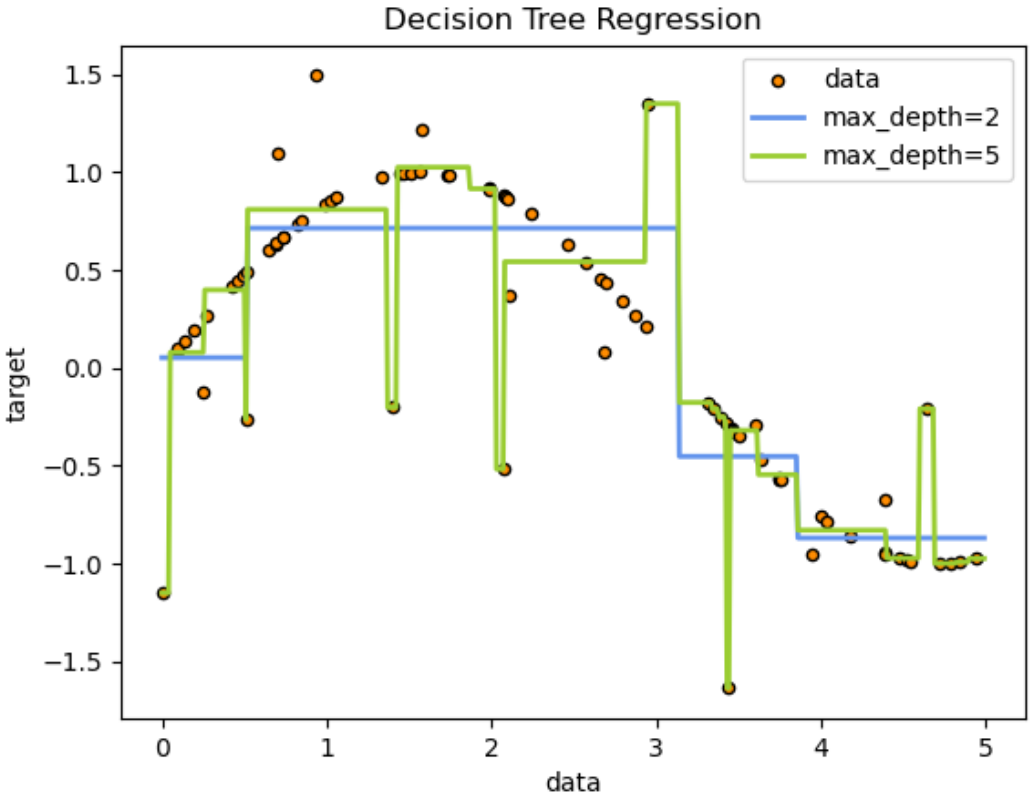
Here is a tabular representation of our dataset.

	Outlook	Temperature	Humidity	Windy	Play Golf
0	Rainy	Hot	High	False	No

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

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.



Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. However scikit-learn implementation does not support categorical variables for now. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

- Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

1.10.1. Classification

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

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

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

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

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

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

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

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

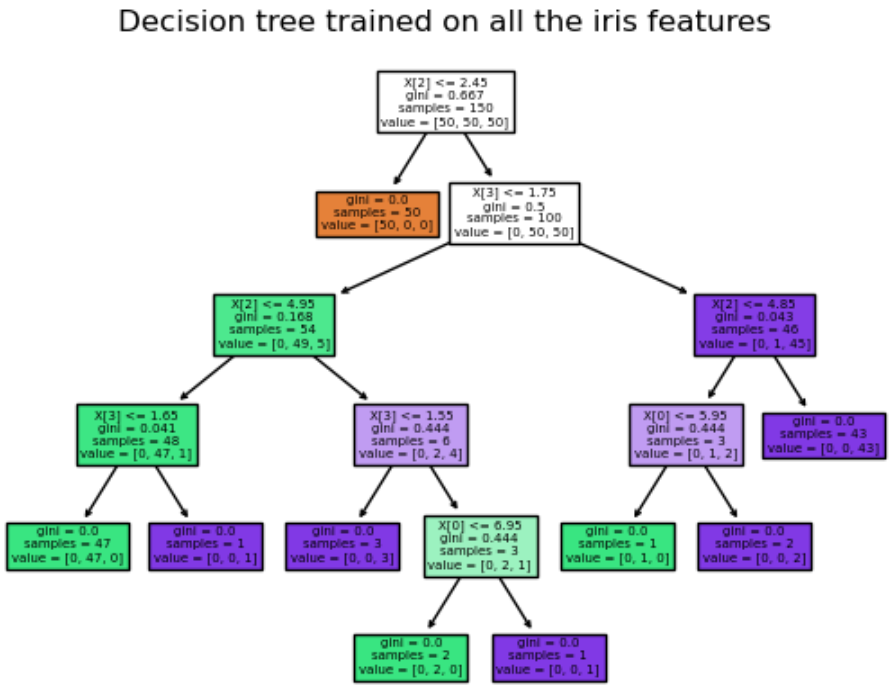
DecisionTreeClassifier is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification.

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

```
>>>
>>>fromsklearn.datasetsimportload_iris
>>>fromsklearnimporttree
>>>iris=load_iris()
>>>X,y=iris.data,iris.target
>>>clf=tree.DecisionTreeClassifier()
>>>clf=clf.fit(X,y)
```

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

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



We can also export the tree in Graphviz format using the export_graphviz exporter. If you use the conda package manager, the graphviz binaries and the python package can be installed with conda install python-graphviz.

Alternatively binaries for graphviz can be downloaded from the graphviz project homepage, and the Python wrapper installed from pypi with pip install graphviz.

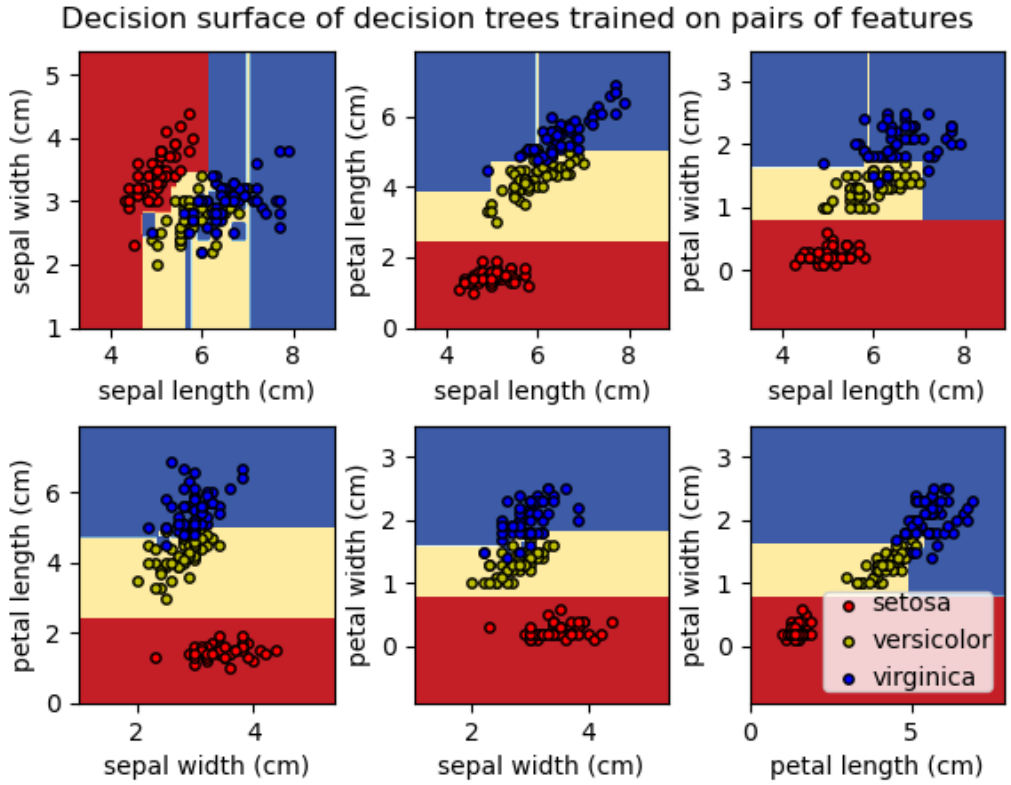
Below is an example graphviz export of the above tree trained on the entire iris dataset; the results are saved in an output file iris.pdf:

```
>>>
>>>importgraphviz
>>>dot_data=tree.export_graphviz(clf,out_file=None)
>>>graph=graphviz.Source(dot_data)
>>>graph.render("iris")
```

The export_graphviz exporter also supports a variety of aesthetic options, including coloring nodes by their class (or value for regression) and using explicit variable and class names if desired. Jupyter notebooks also render these plots inline automatically:

```
>>>
>>>dot_data=tree.export_graphviz(clf,out_file=None,
... feature_names=iris.feature_names,
... class_names=iris.target_names,
```

```
... filled=True,rounded=True,
... special_characters=True)
>>>graph=graphviz.Source(dot_data)
>>>graph
```



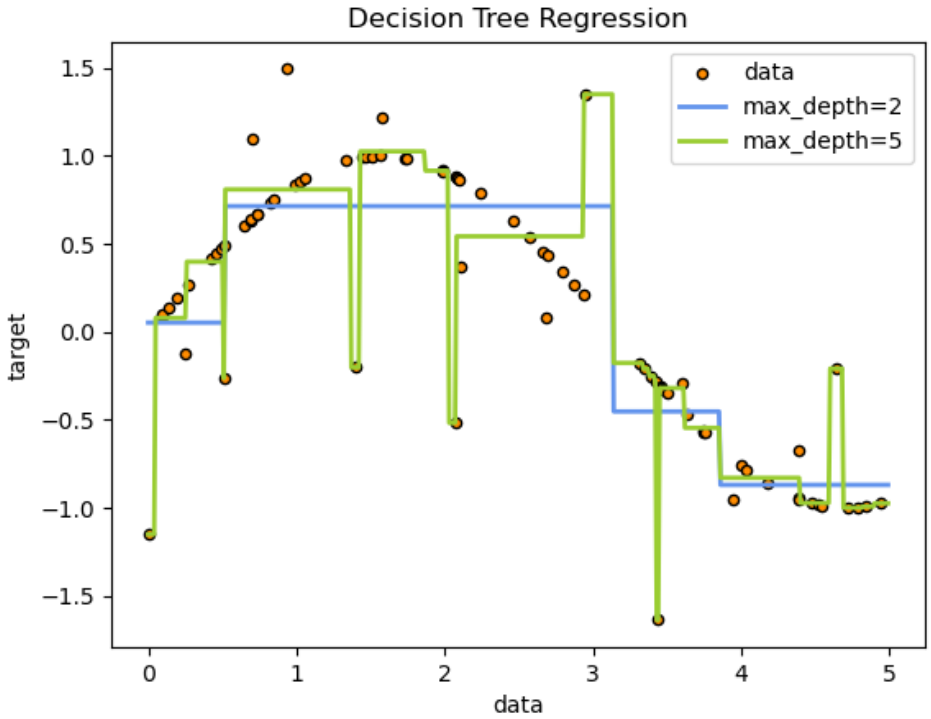
Alternatively, the tree can also be exported in textual format with the function export_text. This method doesn't require the installation of external libraries and is more compact:

```
>>>
>>>fromsklearn.datasetsimportload_iris
>>>fromsklearn.treeimportDecisionTreeClassifier
>>>fromsklearn.treeimportexport_text
>>>iris=load_iris()
>>>decision_tree=DecisionTreeClassifier(random_state=0,max_depth=2)
>>>decision_tree=decision_tree.fit(iris.data,iris.target)
>>>r=export_text(decision_tree,feature_names=iris['feature_names'])
>>>print(r)
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- class: 1
|   |--- petal width (cm) >  1.75
|   |   |--- class: 2
```

Examples:

- Plot the decision surface of decision trees trained on the iris dataset
- Understanding the decision tree structure

1.10.2. Regression



Decision trees can also be applied to regression problems, using the DecisionTreeRegressor class.

As in the classification setting, the fit method will take as argument arrays X and y, only that in this case y is expected to have floating point values instead of integer values:

```
>>>
>>>fromsklearnimporttree
>>>X=[[0,0],[2,2]]
>>>y=[0.5,2.5]
>>>clf=tree.DecisionTreeRegressor()
>>>clf=clf.fit(X,y)
>>>clf.predict([[1,1]])
array([0.5])
```

Examples:

- Decision Tree Regression

1.10.3. Multi-output problems

A multi-output problem is a supervised learning problem with several outputs to predict, that is when Y is a 2d array of shape (n_samples, n_outputs).

When there is no correlation between the outputs, a very simple way to solve this kind of problem is to build n independent models, i.e. one for each output, and then to use those models to independently predict each one of the n outputs. However, because it is likely that the output values related to the same input are themselves correlated, an often better way is to build a single model capable of predicting simultaneously all n outputs. First, it requires lower training time since only a single estimator is built. Second, the generalization accuracy of the resulting estimator may often be increased.

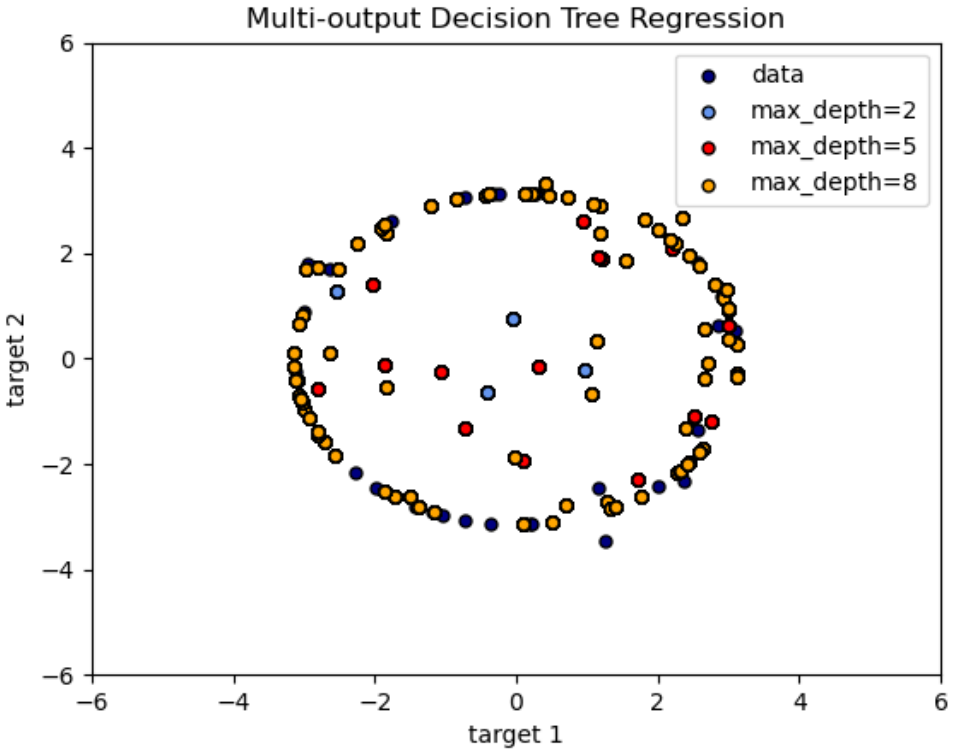
With regard to decision trees, this strategy can readily be used to support multi-output problems. This requires the following changes:

- Store n output values in leaves, instead of 1;
- Use splitting criteria that compute the average reduction across all n outputs.

This module offers support for multi-output problems by implementing this strategy in both DecisionTreeClassifier and DecisionTreeRegressor. If a decision tree is fit on an output array Y of shape (n_samples, n_outputs) then the resulting estimator will:

- Output n_output values upon predict;
- Output a list of n_output arrays of class probabilities upon predict_proba.

The use of multi-output trees for regression is demonstrated in Multi-output Decision Tree Regression. In this example, the input X is a single real value and the outputs Y are the sine and cosine of X.



The use of multi-output trees for classification is demonstrated in Face completion with a multi-output estimators. In this example, the inputs X are the pixels of the upper half of faces and the outputs Y are the pixels of the lower half of those faces.

Face completion with multi-output estimators

PROGRAM NUMBER : 01 Date:15/12/2021

AIM : To implement

(a) Matrix operations (using vectorization),

(b) transformation using python and

(c) SVD using Python.

ALGORITHM

Step 1: Start

Step 2:Create a rank 1 array and print its type

Step 3:Print the shape using shape and print the elements in the array.

Step 4 :Create a rank 2 array and prints shape and the all elements.

Step 5:Create an array of all zeros and print the matrix

Step 6 : Create an array of all ones and print

Step 7 : Create a 2 X 2 identity matrix and print the matrix

Step 8 :Create an array filled with random values

Step 9 :Compute sum of all elements using sum()

Step 10 : Compute sum of each column sum (x, axis=0)

Step 11 : Compute sum of each row sum (x, axis=1)

Step 12 : Compute the matrix dot product using arrange() and dot()

Step 13 : Compute and print outer product of matrix using outer()

Step 14 : Compute elementwise product using multiply()

Step 15 : Stop

Source code

(a) Matrix operations (using vectorization)

```
import numpy as np
a = np.array([1, 2, 3])
print("type: %s" %type(a))
print("shape: %s" %a.shape)
print(a[0], a[1], a[2])
a[0] = 5
print(a)
```

```
b = np.array([[1,2,3],[4,5,6]])
print("\n shape of b:",b.shape)
print(b[0, 0], b[0, 1], b[1, 0])
a = np.zeros((2,2))
print("All zeros matrix:\n  %s" %a)
b = np.ones((1,2))
print("\nAll ones matrix:\n  %s" %b)
d = np.eye(2)
print("\n identity matrix: \n%s"%d)
e = np.random.random((2,2))
print("\n random matrix: \n%s"%e)

 #vectorized sum
print("Vectorized sum example\n")
x = np.array([[1,2],[3,4]])
print("x:\n %s" %x)
print("sum: %s"%np.sum(x))
print("sum axis = 0: %s" %np.sum(x, axis=0))
print(" sum axis = 1: %s" %np.sum(x, axis=1))
#matrix dot product
a = np.arange(10000)
b = np.arange(10000)
dp = np.dot(a,b)
print("Dot product: %s\n" %dp)
#outer product
op = np.outer(a,b)
print("\n Outer product: %s\n" %op)
#elementwise product
ep = np.multiply(a, b)
print("\n Element Wise product: %s \n" %ep)
```

Output :

```
type: <class 'numpy.ndarray'>
shape: 3
1 2 3
[5 2 3]
shape of b: (2, 3)
1 2 4
All zeros matrix:
[[0. 0.]
[0. 0.]]
All ones matrix:
[[1. 1.]]
identity matrix:
[[1. 0.]
[0. 1.]]
random matrix:
[[0.8972642  0.58584636]
[0.05954874 0.71295688]]
```

(b) transformation using python

Algorithm :

Step 1 : Start

Step 2 : Create a rank 2 array

Step 3 : Print the original and transpose matrix

Step 4 : Stop

Source code :

```
import numpy as np
x = np.array([[1,2], [3,4]])
print("Original x: \n%s " %x)
print("\nTranspose of x: \n%s" %x.T)
```

Output :

```
Original x:
[[1 2]
[3 4]]

Transpose of x:
[[1 3]
[2 4]]
```

(c) SVD using Python

Algorithm :

Step 1 : Start

Step 2 : Define the matrix

Step 3 : Print the matrix and print the single value decomposition using svd()

Step 4 : Stop

Source code :

```
# Singular-value decomposition
from numpy import array
from scipy.linalg import svd
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print("A: \n%s" %A)
# SVD
U, s, VT = svd(A)
print("\nU: \n%s" %U)
print("\ns: \n %s" %s)
("\nV^T: \n %s" %VT)
```

Output :

```
A:
[[1 2]
[3 4]
[5 6]]

U:
[[-0.2298477   0.88346102  0.40824829]
[-0.52474482  0.24078249 -0.81649658]
[-0.81964194 -0.40189603  0.40824829]]

s:
[9.52551809 0.51430058]
```

V^T:

[[-0.61962948 -0.78489445]

[-0.78489445 0.61962948]]

PROGRAM NUMBER :02 Date:21/12/2021

AIM: Program to implement hunting exoplanets in space-scatter and line plots.

ALGORITHM :

Step 1: Start

Step 2 : Load exo_train_df dataset

Step 3: Create pandas series for 1st,2nd,3rd and last star and stores it in var star_0,star_1,star_2,star_5086 respectively

Step 4: Create a scatter plot plt for star_0 pandas series

Step 5: Resize plot using 'figure()' function

Step 6: Draw scatter plot between x and y using scatter() function

Step 7: Generate the numbers using arrange() function

Step 8: Display the plot using show()

Step 9: Stop

Source code :

```
from google.colab import drive
drive.mount('/content/drive')

# Load the training dataset.
import pandas as pd
exoTest=pd.read_csv("/content/drive/MyDrive/DSLab/exoTest.csv")
exoTrain=pd.read_csv("/content/drive/MyDrive/DSLab/exoTrain.csv")
exoTrain.head()

# Display Statistical information of the train dataset
exoTrain.describe()

# Check the number of rows and columns in the 'exo_train_df'.
exoTrain.shape

# Find the total number of missing values in the 'exo_train_df'.
exoTrain.isnull().sum()

# Create a Pandas series for the first star and store it in a variable called 'star_0'.
star_0 = exoTrain.iloc[0, :]
star_0.head()

type(star_0)

# Create a Pandas series for the second star and store it in a variable called 'star_1'.
```

```
star_1=exoTrain.iloc[1,:]
star_1.head()

# Create a Pandas series for the third star and store it in a variable called 'star_2'.
star_2=exoTrain.iloc[2,:]
star_2.head()

# Create a Pandas series for the last star and store it in a variable called 'star_5086'.
star_5086=exoTrain.iloc[-1,:]
star_5086.head()

#Create a scatter plot for 'star_0' Pandas series.
# 1. Import the 'numpy' and 'matplotlib.pyplot' modules.
import matplotlib.pyplot as plt
import numpy as np

# 2. Call the 'figure()' function to resize the plot.
plt.figure(figsize=(16, 4))

# Here, 16 means the plot is 16 units wide and 4 units high. Play with these numbers to draw different sized plots.
# Call the 'scatter()' function to make a scatter plot between the x and y values.
# The scatter() function requires two inputs: x and y where x is the data to be plotted on the x-axis and y is the data to be plotted on the y-axis.
# In our case, x is a Pandas series of numbers between 1 and 3197 and y is the 'FLUX' values for a star.
x_values_star_0 = np.arange(1, 3198)
y_values_star_0 = star_0[1:]

# Here, star_0[1:] is a Pandas series containing all the 'FLUX' values starting from the value at index 1 till the value at last index, i.e., 3197
# The 'arange(1, 3198)' function from the 'numpy' module will generate numbers from 1 to 3197.
# 3. Call the 'scatter()' function.
plt.scatter(x_values_star_0, y_values_star_0)

# 4. Call the 'show()' function.
plt.show()
# The 'show()' function displays the plot.
```

Output

	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8	FLUX.9	FLUX.10	FLUX.11	FLUX.12	FLUX.13	FLUX.14	FLUX.15	FLUX.16	FLUX.17	FLUX.18	FLUX.19	FLUX.2
0	2	93.85	83.81	20.10	-26.98	-39.56	-124.71	-135.18	-96.27	-79.89	-160.17	-207.47	-154.88	-173.71	-146.56	-120.26	-102.85	-98.71	-48.42	-86.57	-0.8
1	2	-38.88	-33.83	-58.54	-40.09	-79.31	-72.81	-86.55	-85.33	-83.97	-73.38	-86.51	-74.97	-73.15	-86.13	-76.57	-61.27	-37.23	-48.53	-30.96	-8.1
2	2	532.64	535.92	513.73	496.92	456.45	466.00	464.50	486.39	436.56	484.39	469.66	462.30	492.23	441.20	483.17	481.28	535.31	554.34	562.80	540.1
3	2	326.52	347.39	302.35	298.13	317.74	312.70	322.33	311.31	312.42	323.33	311.14	326.19	313.11	313.89	317.96	330.92	341.10	360.58	370.29	369.7
4	2	-1107.21	-1112.59	-1118.95	-1095.10	-1057.55	-1034.48	-998.34	-1022.71	-989.57	-970.88	-933.30	-889.49	-888.66	-853.95	-800.91	-754.48	-717.24	-649.34	-605.71	-575.6

5 rows × 3198 columns

	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8	FLUX.9	FLUX.10	FLUX.11	FL
count	5087.000000	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.08700
mean	1.007273	1.445054e+02	1.285778e+02	1.471348e+02	1.561512e+02	1.561477e+02	1.469646e+02	1.168380e+02	1.144983e+02	1.228639e+02	1.410852e+02	1.124563e+02	1.05082
std	0.084982	2.150669e+04	2.179717e+04	2.191309e+04	2.223366e+04	2.308448e+04	2.410567e+04	2.414109e+04	2.290691e+04	2.102681e+04	1.942289e+04	1.832810e+04	1.76832
min	1.000000	-2.278563e+05	-3.154408e+05	-2.840018e+05	-2.340069e+05	-4.231956e+05	-5.975521e+05	-6.724046e+05	-5.790136e+05	-3.973882e+05	-2.223300e+05	-2.279016e+05	-2.99675
25%	1.000000	-4.234000e+01	-3.952000e+01	-3.850500e+01	-3.505000e+01	-3.195500e+01	-3.338000e+01	-2.813000e+01	-2.784000e+01	-2.683500e+01	-2.797500e+01	-2.764500e+01	-2.65450
50%	1.000000	-7.100000e-01	-8.900000e-01	-7.400000e-01	-4.000000e-01	-6.100000e-01	-1.030000e+00	-8.700000e-01	-6.600000e-01	-5.600000e-01	-9.600000e-01	-1.310000e+00	-9.10000
75%	1.000000	4.825500e+01	4.428500e+01	4.232500e+01	3.976500e+01	3.975000e+01	3.514000e+01	3.406000e+01	3.170000e+01	3.045500e+01	2.874500e+01	2.703000e+01	2.86750
max	2.000000	1.439240e+06	1.453319e+06	1.468429e+06	1.495750e+06	1.510937e+06	1.508152e+06	1.465743e+06	1.416827e+06	1.342888e+06	1.263870e+06	1.178975e+06	1.08226

8 rows × 3198 columns

(5087, 3198)

```
LABEL      0
FLUX.1     0
FLUX.2     0
FLUX.3     0
FLUX.4     0
  ..
FLUX.3193  0
FLUX.3194  0
FLUX.3195  0
FLUX.3196  0
FLUX.3197  0
Length: 3198, dtype: int64
```

```
LABEL     2.00
FLUX.1   93.85
FLUX.2   83.81
FLUX.3   20.10
FLUX.4  -26.98
Name: 0, dtype: float64
```

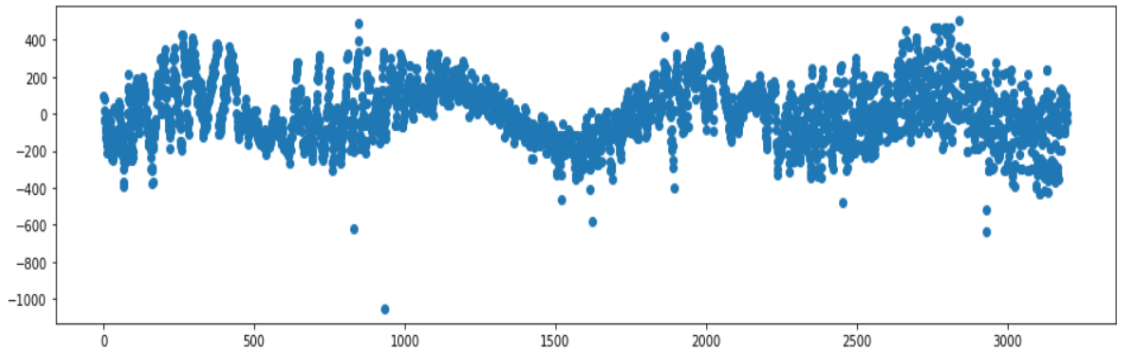
pandas.core.series.Series

```
LABEL     2.00
FLUX.1  -38.88
FLUX.2  -33.83
FLUX.3  -58.54
FLUX.4  -40.09
Name: 1, dtype: float64
```

```
LABEL     2.00
FLUX.1  532.64
FLUX.2  535.92
FLUX.3  513.73
FLUX.4  496.92
Name: 2, dtype: float64
```

```
LABEL     1.00
FLUX.1  323.28
FLUX.2  306.36
FLUX.3  293.16
FLUX.4  287.67
```

Name: 5086, dtype: float64



PROGRAM NUMBER :03 Date:04/01/2022

AIM : Program to implement knn-classification.

Algorithm :

Step 1 : Start

Step 2 : Import the required packages

Step 3: load iris dataset

Step 4: Splitting dataset into training and testing

Step 5 : Preprocessing the iris dataset

Step 6: Create a KNN model and training the model with neighbor value k

Step 7: Testing the KNN model with class labels

Step 8: Find the accuracy of y_test and y_pred

Step 9: Plot the relationship between K and testing accuracy

Step 10: Stop

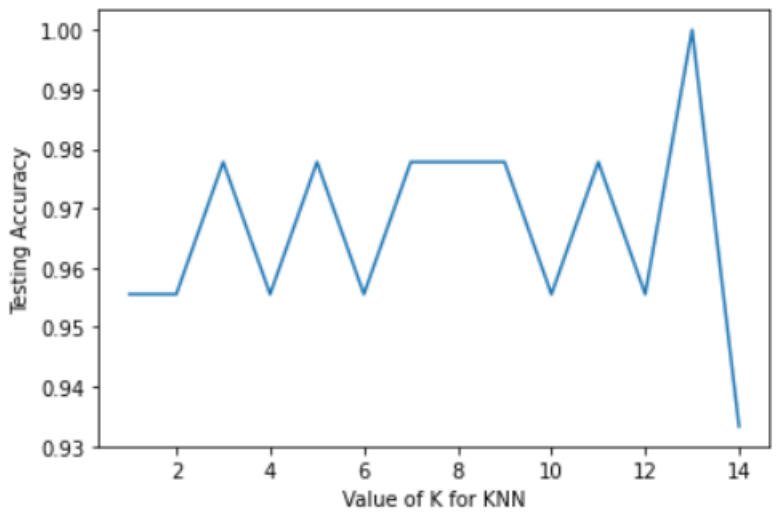
Source code :

```
from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

iris = datasets.load_iris()
X, y = iris.data[:, :], iris.target
Xtrain, Xtest, y_train, y_test = train_test_split(X, y, stratify = y, random_state = 0, train_size = 0.7)

scaler = preprocessing.StandardScaler().fit(Xtrain)
Xtrain = scaler.transform(Xtrain)
Xtest = scaler.transform(Xtest)


Scores = []
k_range = range(1,15)
for k in k_range:

knn = neighbors.KNeighborsClassifier(n_neighbors=k)
 knn.fit(Xtrain, y_train)
  y_pred = knn.predict(Xtest)
scores.append(accuracy_score(y_test, y_pred))
 #print("When  k = %s, accuracy is %s"%(k,accuracy_score(y_test, y_pred)))
```

```
import matplotlib.pyplot as plt

# allow plots to appear within the notebook
%matplotlib inline

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

Output :

```
Text(0, 0.5, 'Testing Accuracy')
```



PROGRAM NUMBER: 04 Date:07/01/2022

AIM**:** Program to implement Naïve Bayes algorithm

ALGORITHM :

Step 1: Start.

Step 2: Import the required data and check the features.

Step 3: Load and print the iris dataset.

Step 4: The dataset is separated by class..

Step 5: Summarize the dataset.

Step 6: Summarize data by class.

Step 7: Introduce Gaussian Probability Density Function.

Step 8: Class probabilities and print confusion matrix.

Step 9: Stop

Source code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
X, y = iris.data[:, :], iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, random_state = 0, train_size = 0.7)
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.naive_bayes import GaussianNB, BernoulliNB, CategoricalNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
scores = []
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores.append(accuracy_score(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
cm
classifier1 = BernoulliNB()
classifier1.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores.append(accuracy_score(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
```

```
cm
#classifier1 = CategoricalNB()
#classifier1.fit(X_train, y_train)
#y_pred = classifier.predict(X_test)
#scores.append(accuracy_score(y_test, y_pred))
#ValueError: Negative values in data passed to CategoricalNB (input X)

print(scores)
```

Output:

```
array([[15,  0,  0],
       [ 0, 15,  0],
       [ 0,  1, 14]])


array([[15,  0,  0],
       [ 0, 15,  0],
       [ 0,  1, 14]])


[0.9777777777777777, 0.9777777777777777]
```

PROGRAM NUMBER: 05 Date:18/01/2022

AIM**:** Program to implement linear and multiple regression techniques using any standard dataset available in the public domain and evaluate its performance.

ALGORITHM :

Step 1: Start

Step 2: Import the required packages

Step 3: Load the diabetes dataset

Step 4: Split the data into training and testing sets

Step 5: Split the targets into training and testing sets

Step 6: Create a linear regression object

Step 7: Train the model using the training sets

Step 8: Make predictions using the testing set

Step 9: Find the mean square error and print coefficient of determination

Step 10: Plots the output

Step 11: Stop

Source code:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
df = datasets.load_diabetes()
df['feature_names']
# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
diabetes_X.shape
diabetes_y.shape
# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
diabetes_X.shape
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
```

```
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
# The coefficients
print("Coefficients: \n", regr.coef_)
 # The mean squared error
 print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test, diabetes_y_pred))
 # The coefficient of determination: 1 is perfect prediction
 print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)
plt.xlabel("age")
plt.ylabel("diabetes progression")
plt.xticks(())
plt.yticks(())

plt.show()
```

Multiple Regression

```
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
diabetes_X.shape
diabetes_X = diabetes_X[:,[0,2]]
diabetes_X.shape
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
# The coefficients
print("Coefficients: \n", regr.coef_)
print("Intercept: \n", regr.intercept_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))
```

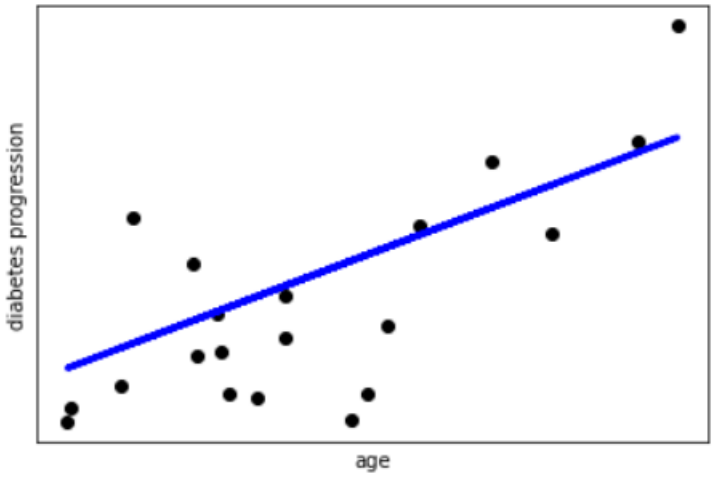
```
x = diabetes_X_test[:, 0]
y = diabetes_X_test[:, 1]
#z = diabetes_X_test[:, 2]
```

The following plots the values in three different angles.

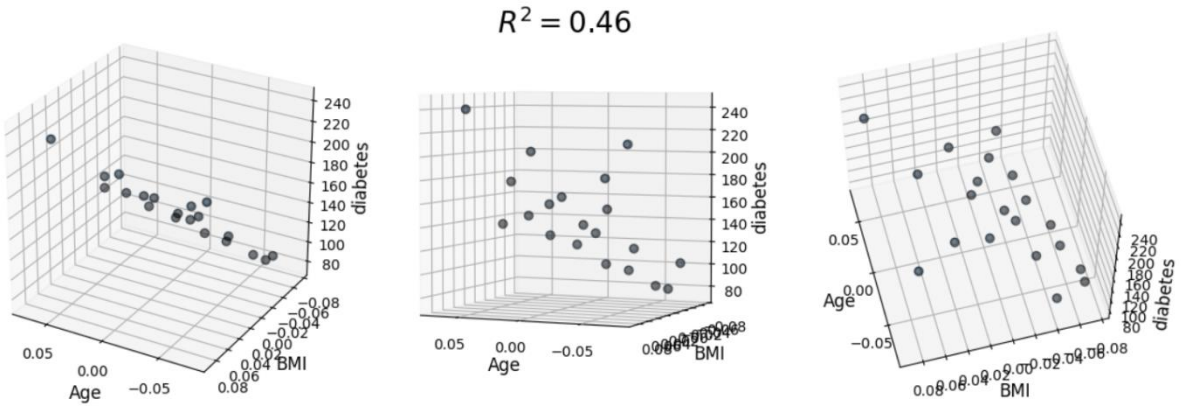
```
plt.style.use('default')

fig = plt.figure(figsize=(12, 4))

ax1 = fig.add_subplot(131, projection='3d')
ax2 = fig.add_subplot(132, projection='3d')
ax3 = fig.add_subplot(133, projection='3d')

axes = [ax1, ax2, ax3]

for ax in axes:
  ax.plot(x, y, diabetes_y_pred, color='k', zorder=15, linestyle='none', marker='o', alpha=0.5)
 ax.scatter(x.flatten(), y.flatten(), diabetes_y_pred, facecolor=(0,0,0,0), s=20, edgecolor='#70b3f0')
 ax.set_xlabel('Age', fontsize=12)
 ax.set_ylabel('BMI', fontsize=12)
  ax.set_zlabel('diabetes', fontsize=12)
 ax.locator_params(nbins=4, axis='x')
 ax.locator_params(nbins=5, axis='x')

 ax1.view_init(elev=28, azim=120)
 ax2.view_init(elev=4, azim=114)
 ax3.view_init(elev=60, azim=165)

 fig.suptitle('$R^2 = %.2f$' % r2_score(diabetes_y_test, diabetes_y_pred), fontsize=20)

 fig.tight_layout()
```

Output:

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']

(442, 10)
(442,)
(442, 1)
Coefficients:
[938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```



(442, 2)
Coefficients:
[139.20420118 912.45355549]
Intercept:
152.8767000140564
Mean squared error: 2596.60
Coefficient of determination: 0.46



PROGRAM NUMBER: 06 Date:28/01/2022

AIM**:** Program to implement text classification using Support Vector Machine.

Algorithm :

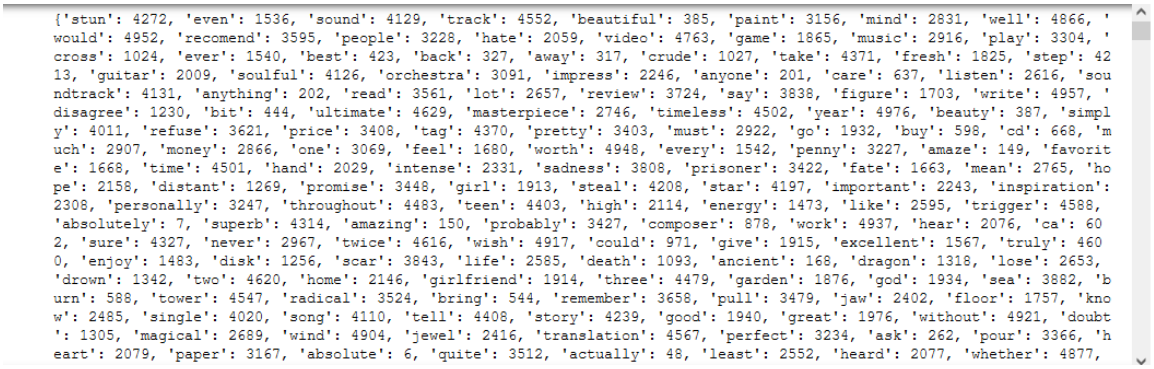
Step 1: Start
Step 2: Import the required packages
Step 3: Load the Corpus dataset
Step 4: Remove blank rows if any
Step 5: Change all test to lowercase
Step 6: Tokenization is done on dataset
Step 7: By using wordNetLammatizer pos tags to understand if the word is noun or verb/adjective etc
Step 8: Train the model using training sets
Step 9: Fit the training dataset on the classifier
Step 10: Predict the labels on validation dataset
Step 11: By using accuracy_score function find the accuracy
Step 12: Stop

Source code :

```
import pandas as pd
import numpy as np
from nltk.tokenize import word_tokenize
from nltk import pos_tag
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
from nltk.stem import WordNetLemmatizer
from sklearn.preprocessing import LabelEncoder
from collections import defaultdict
from nltk.corpus import wordnet as wn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import model_selection, naive_bayes, svm
from sklearn.metrics import accuracy_score

np.random.seed(500)

Corpus = pd.read_csv(r"corpus.csv",encoding='latin-1')

# Step - a : Remove blank rows if any.
Corpus['text'].dropna(inplace=True)
# Step - b : Change all the text to lower case. This is required as python interprets 'dog' and
'DOG' differently
Corpus['text'] = [entry.lower() for entry in Corpus['text']]
# Step - c : Tokenization : In this each entry in the corpus will be broken into set of words
```

```
Corpus['text']= [word_tokenize(entry) for entry in Corpus['text']]
# Step - d : Remove Stop words, Non-Numeric and perfom Word Stemming/Lemmenting.
# WordNetLemmatizer requires Pos tags to understand if the word is noun or verb or adjective etc. By
default it is set to Noun
tag_map = defaultdict(lambda : wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV
for index,entry in enumerate(Corpus['text']):
  # Declaring Empty List to store the words that follow the rules for this step
 Final_words = []
  # Initializing WordNetLemmatizer()
  word_Lemmatized = WordNetLemmatizer()
 # pos_tag function below will provide the 'tag' i.e if the word is Noun(N) or
Verb(V) or something else.
for word, tag in pos_tag(entry):
  # Below condition is to check for Stop words and consider only alphabets
 if word not in stopwords.words('english') and word.isalpha():
word_Final = word_Lemmatized.lemmatize(word,tag_map[tag[0]])
Final_words.append(word_Final)
# The final processed set of words for each iteration will be stored in 'text_final'
Corpus.loc[index,'text_final'] = str(Final_words)

Train_X, Test_X, Train_Y, Test_Y = model_selection.train_test_split(Corpus['text_final'],
Corpus['label'],test_size=0.3)

Encoder = LabelEncoder()
Train_Y = Encoder.fit_transform(Train_Y)
Test_Y = Encoder.fit_transform(Test_Y)

Tfidf_vect = TfidfVectorizer(max_features=5000)
Tfidf_vect.fit(Corpus['text_final'])
Train_X_Tfidf = Tfidf_vect.transform(Train_X)
Test_X_Tfidf = Tfidf_vect.transform(Test_X)

print(Tfidf_vect.vocabulary_)

print(Train_X_Tfidf)


# Classifier - Algorithm - SVM
# fit the training dataset on the classifier
SVM = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
SVM.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
predictions_SVM = SVM.predict(Test_X_Tfidf)
# Use accuracy_score function to get the accuracy
print("SVM Accuracy Score -> ",accuracy_score(predictions_SVM, Test_Y)*100)
```

Output :



(0, 3658) 0.2896999547088821

(0, 3561) 0.29449641491430995

(0, 2922) 0.229683025366997

(0, 1940) 0.13406125327954532

(0, 1536) 0.17761496997588844

(0, 517) 0.321056290554803

(0, 488) 0.12303572865008613

(0, 238) 0.2448559358109696

(1, 4687) 0.21384275526442909

(1, 4069) 0.3566872275481094

(1, 3434) 0.21279175847748263

(1, 3319) 0.8157357261127677

(1, 2595) 0.2173336717856602

(1, 1252) 0.2074693534878867

(1, 598) 0.1614401835472762

(2, 4734) 0.21251405574612364

(2, 4621) 0.17383471522304228

(2, 4464) 0.11898591577849023

(2, 4197) 0.13515537469996092

: :

(6998, 2522) 0.11512409752599596

(6998, 2130) 0.13650214385741868

(6998, 1976) 0.07126908030410523

(6998, 1788) 0.22013355385880556

(6998, 1755) 0.19935027840675415

(6998, 1719) 0.13508979239544552

(6998, 1595) 0.13521530232348064

(6998, 1579) 0.1852028677205329

(6998, 1545) 0.13386451534221278

(6998, 1540) 0.09899700620855782

(6998, 1299) 0.2957222991515454

(6998, 1186) 0.2328691080708402

(6998, 488) 0.36187164406935557

(6999, 4866) 0.16162053232828144

(6999, 4149) 0.5217385422290042

(6999, 3823) 0.3649735868975402

(6999, 2902) 0.16568268411504228

(6999, 2699) 0.14615260074275177

(6999, 1976) 0.14155242939037596

(6999, 1672) 0.28173977593289695

(6999, 1536) 0.17292898689607455

(6999, 1406) 0.2877416804485293

(6999, 1238) 0.314256617088249

(6999, 319) 0.28499442233774624

(6999, 50) 0.3571519291530235

SVM Accuracy Score -> 84.7

PROGRAM NUMBER: 07 Date:10/02/2022

AIM**:** Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.

Algorithm :

Step 1: Start
Step 2: Import the required data and check the features
Step 3: Load and print the target names and features of iris dataset.
Step 4: Set default criterion as Gini
Step 5: Print the accuracy on using Gini
Step 6: Change criterion to entropy
Step 7: Print the accuracy data on using entropy
Step 8: Change criterion to entropy with min_samples_split to 50. Default value is 2.
Step 9: Print the accuracy
Step 10: Visualize the decision tree
Step 11: Stop

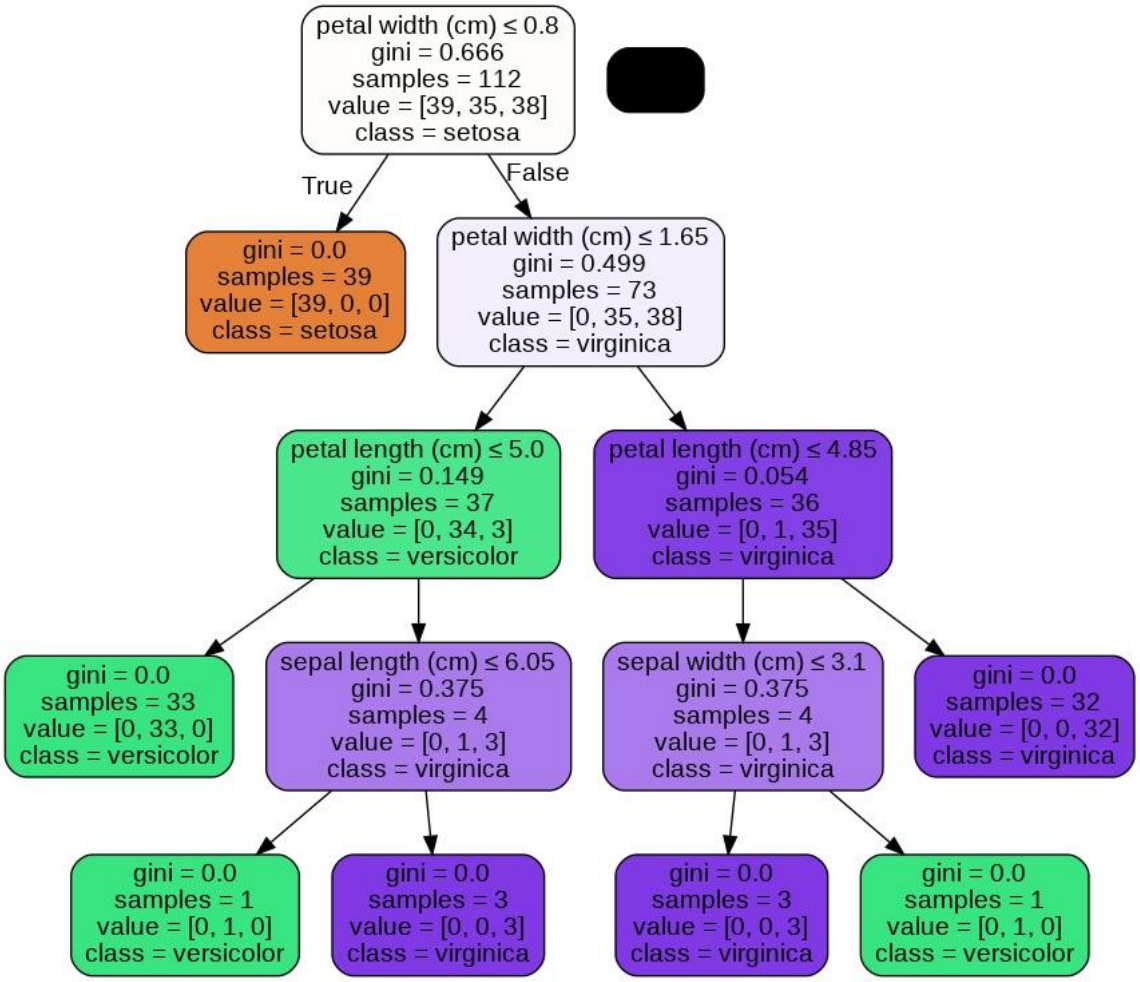
Source code:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

#load iris data
data = load_iris()
data.data.shape
print('classes to predict: ',data.target_names)
print('Features: ',data.feature_names)
X = data.data
y = data.target

display (X.shape, y.shape)
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,random_state = 50, test_size = 0.25)
#default criterion is Gini
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import accuracy_score
print('Accuracy on train data using Gini: ',accuracy_score(y_true = y_train, y_pred = classifier.predict
(X_train)))
print('Accuracy on test data using Gini: ',accuracy_score(y_true = y_test, y_pred = y_pred))
```

```
#change criterion to entropy
classifier_entropy = DecisionTreeClassifier(criterion='entropy')
classifier_entropy.fit(X_train, y_train)
y_pred_entropy = classifier_entropy.predict(X_test)
print('Accuracy on train data using entropy', accuracy_score(y_true=y_train, y_pred =
classifier_entropy.predict(X_train)))
print('Accuracy on test data using entropy', accuracy_score(y_true=y_test, y_pred = y_pred_entropy))
#change criterion to entropy with min_samples_split to 50. Default value is 2
classifier_entropy1 = DecisionTreeClassifier(criterion='entropy', min_samples_split=50)
classifier_entropy1.fit(X_train, y_train)
y_pred_entropy1 = classifier_entropy1.predict(X_test)
print('Accuracy on train data using entropy', accuracy_score(y_true=y_train, y_pred =
classifier_entropy1.predict(X_train)))
print('Accuracy on test data using entropy', accuracy_score(y_true=y_test, y_pred = y_pred_entropy1))
#visualise the decision tree

from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
#the students can try using classifier, classifier_entropy and classifier_entropy1
#as first parameter below.
export_graphviz(classifier, out_file = dot_data,filled = True, rounded = True,
special_characters = True, feature_names = data.feature_names, class_names = data.target_names)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Output:

```
(150, 4)

 classes to predict:  ['setosa' 'versicolor' 'virginica']
Features:  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

  (150, 4)
   (150,)

Accuracy on train data using Gini:  1.0
Accuracy on test data using Gini:  0.9473684210526315

Accuracy on train data using entropy 1.0
 Accuracy on test data using entropy 0.9473684210526315

Accuracy on train data using entropy 0.9642857142857143
 Accuracy on test data using entropy 0.9473684210526315
```


petal width (cm) ≤ 0.8
gini = 0.666
samples = 112
value = [39, 35, 38]
class = setosa
True
False
gini = 0.0
samples = 39
value = [39, 0, 0]
class = setosa
petal width (cm) ≤ 1.65
gini = 0.499
samples = 73
value = [0, 35, 38]
class = virginica
petal length (cm) ≤ 5.0
gini = 0.149
samples = 37
value = [0, 34, 3]
class = versicolor
petal length (cm) ≤ 4.85
gini = 0.054
samples = 36
value = [0, 1, 35]
class = virginica
gini = 0.0
samples = 33
value = [0, 33, 0]
class = versicolor
sepal length (cm) ≤ 6.05
gini = 0.375
samples = 4
value = [0, 1, 3]
class = virginica
sepal width (cm) ≤ 3.1
gini = 0.375
samples = 4
value = [0, 1, 3]
class = virginica
gini = 0.0
samples = 32
value = [0, 0, 32]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor