



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

M.TECH (INTEGRATED) SOFTWARE ENGINEERING 2020-21

Soft Computing -SWE1011

Slot: C1+TC1

Project Component

Final Review

Title: Football Match Prediction

Using MLP, Logistic Regression and SVC

Submitted by,

B SUDHARSHAN - 18MIS0273

T HARISHKUMAR – 18MIS0358

Submitted to,

SUBHASHINI.R

Abstract:

Predicting the results of football matches poses an interesting challenge due to the fact that the sport is so popular and widespread. Predicting the results of sports matches is interesting to many, from fans to punters. It is also interesting as a research problem, in part due to its difficulty, because the result of a sports match is dependent on many factors, such as the morale of a team (or a player), skills, current score, etc. So even for sports experts, it is very hard to predict the exact results of sports matches. So it can be vastly unpredictable at times; it may be safe to say that it is one of the most unpredictable sports ever. Having said that there also lies a possibility of predicting a match of an England Premiere League based on their previous records against each other that is what we will be exploring in this project.

Introduction:

In this football Prediction we have used three classifiers model to predict the exact score taken in the Particular league match Dataset and also we compare the accuracy of the Predicted output with three models. In this Prediction we have used the past 1 decade matches in the particular league (England Premiere League).

We have implemented with 3 neural network models to predict the exact score of the matches.

Our Input file is in csv form. The Models are

1. Multi-layer Perceptron
2. Logistic Regression
3. Support Vector Machine

Software Used:

Language - Python

Tool - Jupyter Notebook (anaconda 3)

Dataset:

HTGD - Home team goal difference

ATGD - away team goal difference

HTP - Home team points

ATP - Away team points

DiffFormPts - Difference in points.

DiffLP - Difference in last year's prediction

Literature Review:

Study	Division	Features	Future work/ Limitation	Accuracy
1. Arabzad et al., 2014	Iran Pro League	Teams, form of teams in last matches & league, quality of last opponents	Distance between matches, Club investment, Weathe	N.A.
2. McCabe and Trevathan, 2008	Premier League	Points for and against, win loss record, home and away Performance, performance in previous 4 games, ranking, location, player availability	Richer feature sets	54%
3. Huang and Chang, 2010	World Cup 2006	Goals for, Shots, Shots on Goal, Corner Kicks, Free kicks, Ball possession & Fouls	Limited training data	77%
4. Tax and Joustra, 2015	Dutch Eredivisie	Goals for, goals against, result previous matched, top scorers, days	Expand to other Leagues	55%

		since previous match, win/draw/lose percentage, Odds		
5. Aslan and Inceoglu, 2007	Italian Serie A	Home rating & Away rating for home team and away team	Different leagues, structures and input features	51%
6. Aslan and Inceoglu, 2008	Italian Serie A	Home Rating home team, Away Rating away team	Different leagues, structures and input features	53%

Some More brief review of articles :

7. Multilayer Perceptron for Prediction of 2006 World Cup Football game.

Author : Kou-Yuan Huang and Kai-Ju Chen

Multilayer perceptron (MLP) with back-propagation learning rule is adopted to predict the winning rates of two teams according to their official statistical data of 2006 World Cup Football Game at the previous stages. There are training samples from three classes: win, draw, and loss. At the new stage, new training samples are selected from the previous stages and are added to the training samples, then They retrain the neural network. It is a type of on-line learning. The 8 features are selected with ad hoc choice. They use the theorem of Mirchandani and Cao to determine the number of hidden nodes. And after the testing in the learning convergence, the MLP is determined as 8-2-3 model. The learning rate and momentum coefficient are determined in the cross-learning. The prediction accuracy achieves 75% if the draw games are excluded. In this study, They adopt multilayer perceptron with back-propagation learning to predict the winning rate of 2006 WCFG. They select 8 significant statistical records from 17 official records of 2006 WCFG. The 8 records of each team are transformed into relative ratio values with

another team. Then the average ratio values of each team at previous stages are fed into 8-2-3 MLP for predicting the win and loss. The teams of 3 wins, 3 losses, and draws at stage 1 are selected as the training samples. The 8 records and training samples are selected by ad hoc choice. It is a common process to the real application of an algorithm. New training samples are added to the training set of the previous stages, and then we retrain the neural network. It is a type of on-line learning. The learning rate and the momentum coefficient are determined by the less average and deviation time in the cross-learning.

They use the theorem of Mirchandani and Cao to determine the number of hidden nodes. It is 6, and the MLP model is 8-6-3. After the testing in the learning convergence, the MLP is determined as 8-2-3 model. They can infer that some training samples are grouped together and we do not need the 6 hidden nodes.

If the draw games are excluded, the prediction accuracy can achieve 75% .

8. Investigation of Naive Bayes Combined with Multilayered Perceptron for Arabic Sentiment Analysis and Opinion Mining.

Author : Shakir Mrayyen, Mohammad Subhi Al-Batah, Malek Alzaqebah

The sentiment classification is a task of classifying the people's text of reviews by performing Text Analysis techniques and Natural Language Processing to determine whether the people's attraction about some topic (product, hotel, or movie, etc.) is positive, negative, or neutral.

Sentiment analysis and opinion mining recently become very popular and active research, that due to the importance of the sentiment analysis in various fields, such as in politic, marketing, education, healthcare etc.

In addition, sentiment analysis plays an important role in making decisions for individuals, organizations or governments; an appropriate decision can be made wisely by getting the opinion from others. Many researchers have been studied the sentiment analysis in terms of English language, nonetheless the research work in terms of Arabic language still insufficient. Therefore, more research work of Arabic is required. The authors Farraetal in used sentiment mining of Arabic text at both sentence level and document level. The study works in a grammatical approach and a semantic approach for sentence-level sentiment mining in Arabic text, and a document-level, also they construct semantic dictionary includes Arabic roots.

The results showed high accuracy in grammatical approach. El Beltagy et al. studied the possibility of determining the semantic orientation in terms of Arabic Egyptian. The authors used two datasets from tweeter, first datasets contained 500 tweets, classified into 155 positive tweets, 310 negative tweets, and 35 neutral tweets; second dataset is the Dostour dataset which has 100 random comments. The comments were classified 40 positive, 38 negative, and 22 neutral.

9. Football Result Prediction by Deep Learning Algorithm :

Author : Stefan Samba

Deep learning has shown promising results in classification and prediction of complex cases. During this study, football or soccer, hereafter named football, will be the sport representing this complex area. Football is a decent representation of a complex area because of its dependency on many features, such as team strategy, player morale, team morale, quality of offense, quality of defence, possession and fatigue and luck. For laws of the game, Fifa (2018) can be consulted. Large and detailed collections of historical data, per season, per league and per match offers great possibilities to investigate the modelling of football results. The remainder of this section contains an introduction to football result prediction, the research question and the outline of this paper.

Football result prediction shows its relevance in a practical and scientific way. From a practical point of view it is interesting for clubs and nations to improve performance, for bookmakers to increase finances and in other ways for different football related parties. From a scientific point of view, it is interesting because the outcome of matches is difficult due to its dependency of many features (Aslan & Inceoglu, 2007; Bunker & Thabtah, 2019; Langaroudi & Yamaghani, 2019) and luck (Aoki et al., 2017).

The multilayer perceptron (MLP) is a deep learning model that is able to separate non-linear data. As indicated by its name, the MLP consists of perceptrons within multiple layers. More information about the perceptron and its components can be found in Minsky and Papert (1988). The concept of a MLP finds its roots in the biological brain where each neuron can receive, process and transmit incoming signals. Combining the strength of the different neurons together in an artificial settings leads to perceptron with multiple layers. These layers and how the neurons within the layers connect to each other is considered the architecture of the network (Goodfellow et al., 2016)

10. Support Vector Machine – Based Prediction system for a football match Result.

Author : Chinwe Peace Igiri

Predictive models recently have been employed to predict the weather, student performance, and stock market fluctuations. The use of machine learning and data-mining techniques to improve prediction accuracy has yielded positive results in the aforementioned fields. Consequently, it would not be out of place to apply the same techniques to football. Due to the contemporary popularity of sports, many organizations have invested a great deal to obtain better results in predicting football matches; accordingly, the prediction of game results has become an area of interest.

Data mining, a widely accepted method to predict and explain events, is an appropriate tool for this purpose. Various data mining techniques have been employed to predict game results in recent years, such as artificial neural networks, decision trees, Bayesian method, logistic regression, and support vector machines (SVM) and fuzzy methods. This study seeks to study the effect of applying a system based on a support vector machine (SVM) to predict the results of football matches. The football result prediction system is a very broad area of study in computing, economics and business. For the purpose of this research, this system will be developed using data mining tools through knowledge discovery in databases (KDD).

The emphasis will be on implementing the system using a SVM. Nonetheless, other related work on prediction systems will be reviewed for the purpose of completeness.

Methodology:

We have implemented with 3 neural network models to predict the exact score of the matches.

Our Input file is in csv form. The Models are

1. Multi-layer Perceptron
2. Logistic Regression
3. Support Vector Machine

Multi-layer Perceptron:

A Multilayer Perceptron is a Perceptron with multiple layer. It has 3 layers

- Input layer,
- Hidden layer
- Output layer

Each Neurons computes an Activation function.

Input Layer :

1. Introduces input values into the Network.
2. No activation function or other Processing

Hidden Layer :

1. Performs classification of features.
2. Two hidden layers are sufficient to solve any Problem.
3. More Hidden layers can make More accuracy.

Output Layer :

1. Functionally just like Hidden Layer.
2. Output are passed to the world.
3. If the output layer is wrong we will change the Weights until it reaches correct Output.

Steps in Multi-layer Perceptron:

Perceptron Training Algorithm is Used.

Step 0: initialize weights, bias, learning rate($0 < a \leq 1$)

Step 1: perform step 2-6 until final stopping condition is false

Step 2: perform steps 3-5 for each bipolar or binary training pair indicated by s:t

Step 3: input layer is applied with identity activation fn: $x_i = s_i$

Step 4: calculate output response of each input $j=1$ to m first, net input is calculated

$$y_{in} = b + \sum_{i=1}^n x_i w_i$$

Activation are applied over the net input to calculate the output response

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \leq y_{in} \leq \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

Step 5: Make adjustment in weight and bias $j=1$ to m and $i=1$ to n

If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha x_i$$

$$b(\text{new}) = b(\text{old}) + \alpha t$$

else, we have

$$w_i(\text{new}) = w_i(\text{old})$$

$$b(\text{new}) = b(\text{old})$$

Step 6: Test for the stopping condition. If there no change in weight then stop the training process else start again from step2.

Logistic Regression:

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes. In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no). Mathematically, a logistic regression model predicts $P(Y=1)$ as a function of X . It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

Types of Logistic Regression:

Generally, logistic regression means binary logistic regression having binary target variables, but there can be two more categories of target variables that can be predicted by it. Based on those number of categories, Logistic regression can be divided into following types –

Binary or Binomial:

In such a kind of classification, a dependent variable will have only two possible types either 1 and 0. For example, these variables may represent success or failure, yes or no, win or loss etc.

Multinomial:

In such a kind of classification, dependent variable can have 3 or more possible unordered types or the types having no quantitative significance. For example, these variables may represent “Type A” or “Type B” or “Type C”.

Ordinal:

In such a kind of classification, dependent variable can have 3 or more possible ordered types or the types having a quantitative significance. For example, these variables may represent “poor” or “good”, “very good”, “Excellent” and each category can have the scores like 0,1,2,3.

Binary Logistic Regression model:

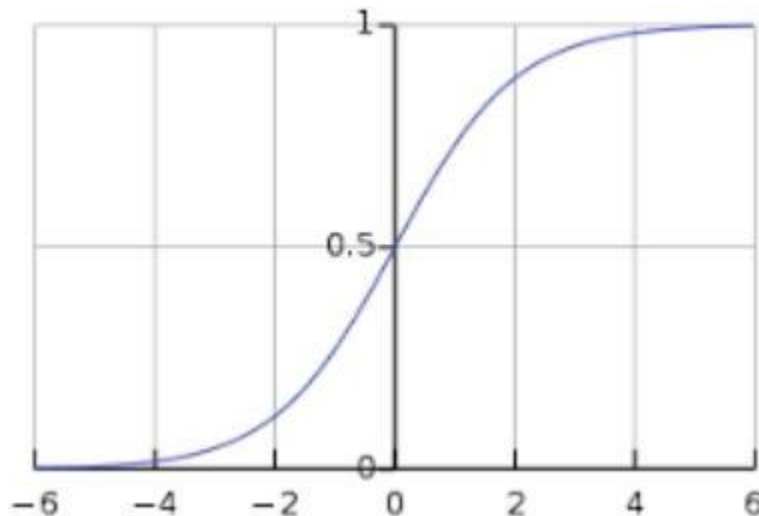
The simplest form of logistic regression is binary or binomial logistic regression in which the target or dependent variable can have only 2 possible types either 1 or 0. It allows us to model a relationship between multiple predictor variables and a binary/binomial target variable. In case of logistic regression, the linear function is basically used as an input to another function such as g in the following relation-

$$h\theta(x) = g(\theta^T x) \text{ where } 0 \leq h\theta \leq 1$$

Here, g is the logistic or sigmoid function which can be given as follows –

$$g(z) = \frac{1}{1 + e^{-z}} \text{ where } z = \theta^T x$$

The sigmoid curve can be represented with the help of following graph. We can see the values of y-axis lie between 0 and 1 and crosses the axis at 0.5.



The classes can be divided into positive or negative. The output comes under the probability of positive class if it lies between 0 and 1. For our implementation, we are interpreting the output of hypothesis function as positive if it is ≥ 0.5 , otherwise negative.

We also need to define a loss function to measure how well the algorithm performs using the weights on functions, represented by theta as follows – $h = (X\theta)$

$$J(\theta) = \frac{1}{m} (-y^T \log(h) - (1-y)^T \log(1-h))$$

Now, after defining the loss function our prime goal is to minimize the loss function. It can be done with the help of fitting the weights which means by increasing or decreasing the weights. With the help of derivatives of the loss function w.r.t each weight, we would be able to know what parameters should have high weight and what should have smaller weight.

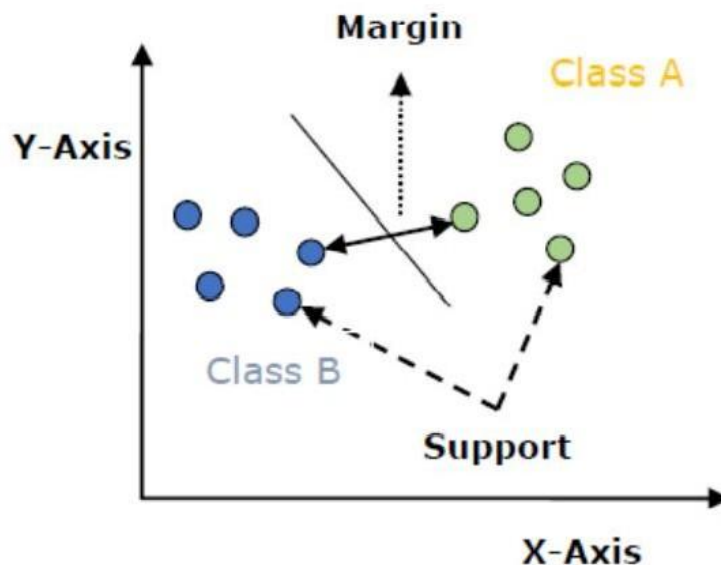
The following gradient descent equation tells us how loss would change if we modified the parameters – $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} X^T (g(X\theta) - y)$

Support Vector Machine:

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

Working of SVM:

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



The followings are important concepts in SVM –

- **Support Vectors** – Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.
- **Hyperplane** – As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes

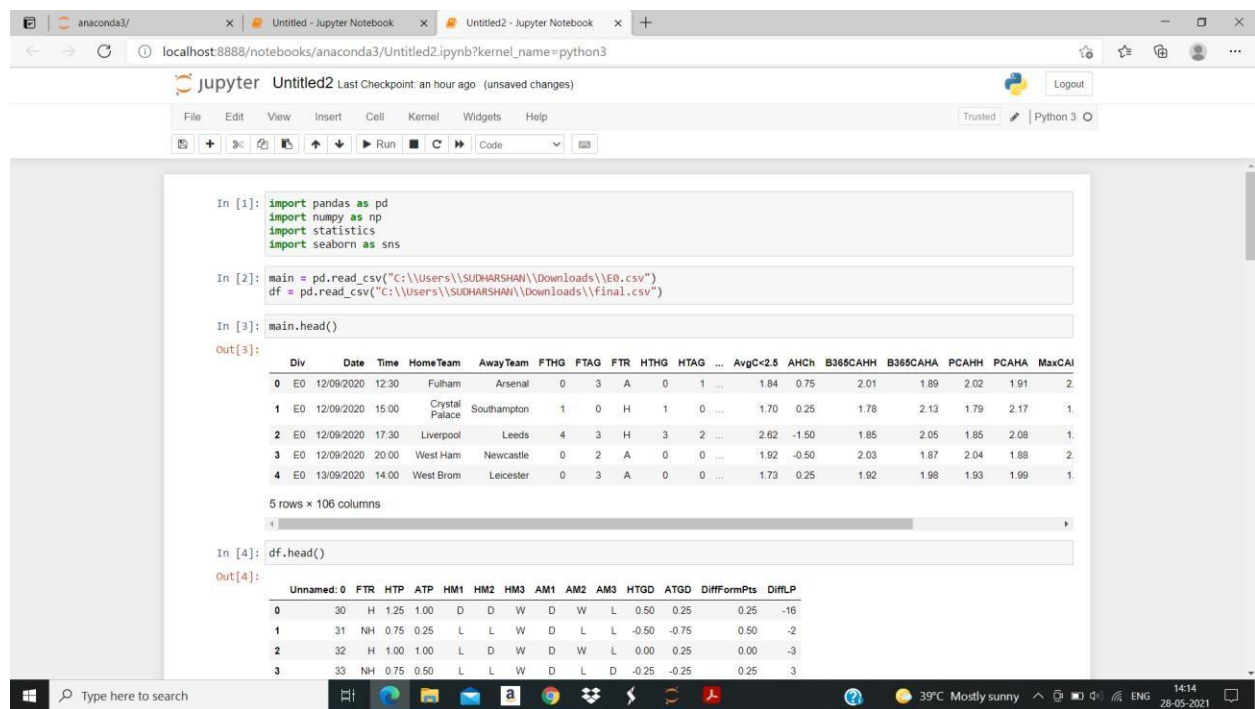
- **Margin** – It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

The main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps –

- First, SVM will generate hyperplanes iteratively that segregates the classes in best way.
- Then, it will choose the hyperplane that separates the classes correctly.

50% Implementation:

Importing packages and csv files:



```
In [1]: import pandas as pd
import numpy as np
import statistics
import seaborn as sns

In [2]: main = pd.read_csv("C:\\Users\\SUDHARSHAN\\Downloads\\E0.csv")
df = pd.read_csv("C:\\Users\\SUDHARSHAN\\Downloads\\final.csv")

In [3]: main.head()
Out[3]:
```

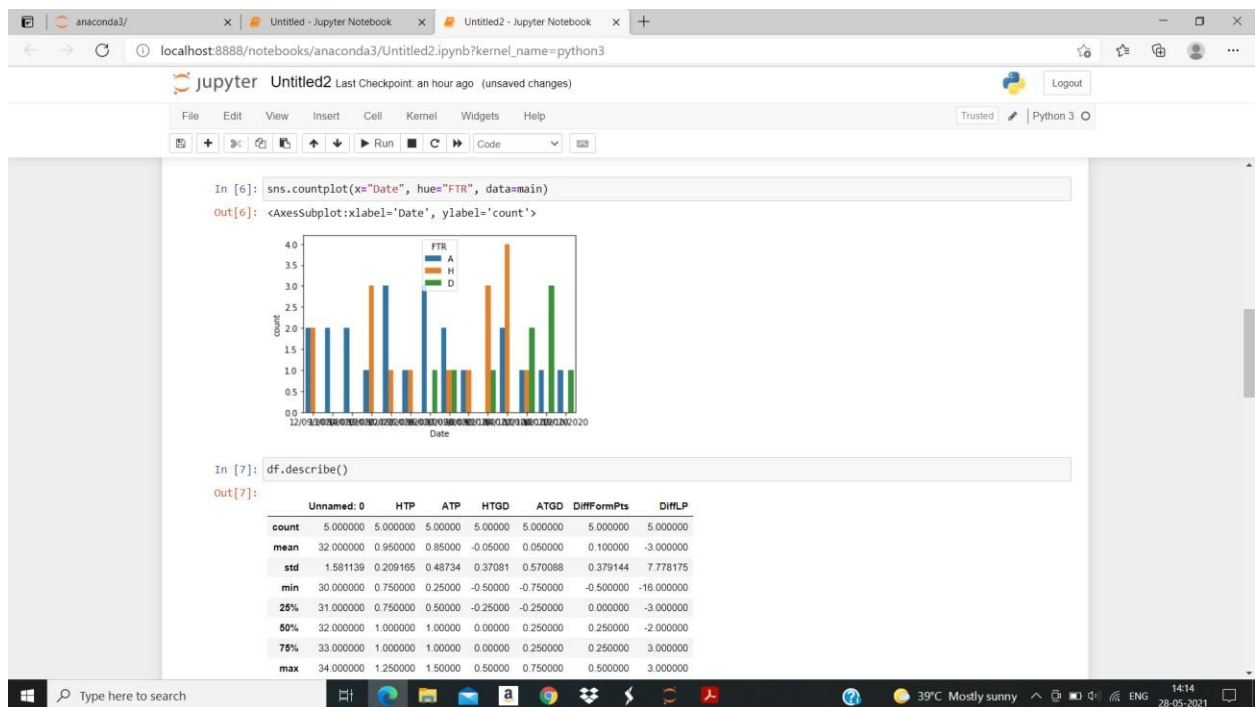
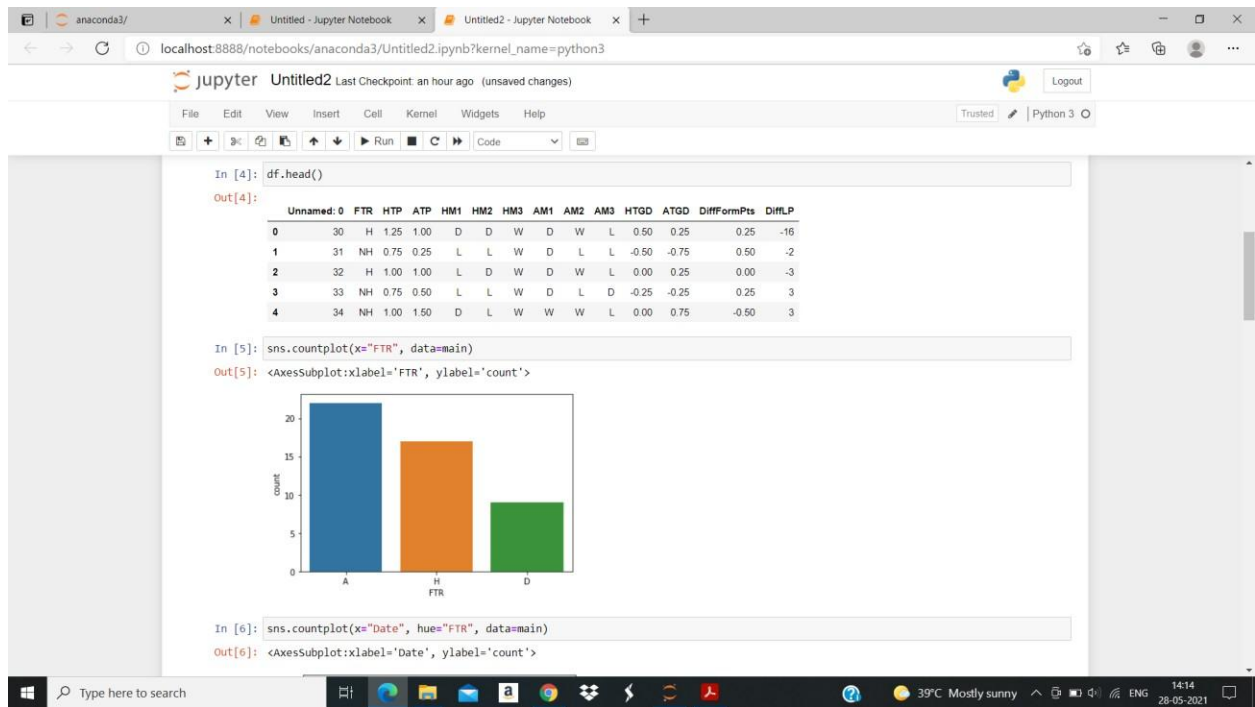
	Div	Date	Time	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	...	AvgC<2.5	AHCh	B365CAHH	B365CAHA	PCAHH	PCAHA	MaxCAI
0	E0	12/09/2020	12:30	Fulham	Arsenal	0	3	A	0	1	...	1.84	0.75	2.01	1.89	2.02	1.91	2.
1	E0	12/09/2020	15:00	Crystal Palace	Southampton	1	0	H	1	0	...	1.70	0.25	1.78	2.13	1.79	2.17	1.
2	E0	12/09/2020	17:30	Liverpool	Leeds	4	3	H	3	2	...	2.62	-1.50	1.85	2.05	1.85	2.08	1.
3	E0	12/09/2020	20:00	West Ham	Newcastle	0	2	A	0	0	...	1.92	-0.50	2.03	1.87	2.04	1.88	2.
4	E0	13/09/2020	14:00	West Brom	Leicester	0	3	A	0	0	...	1.73	0.25	1.92	1.98	1.93	1.99	1.

5 rows x 106 columns

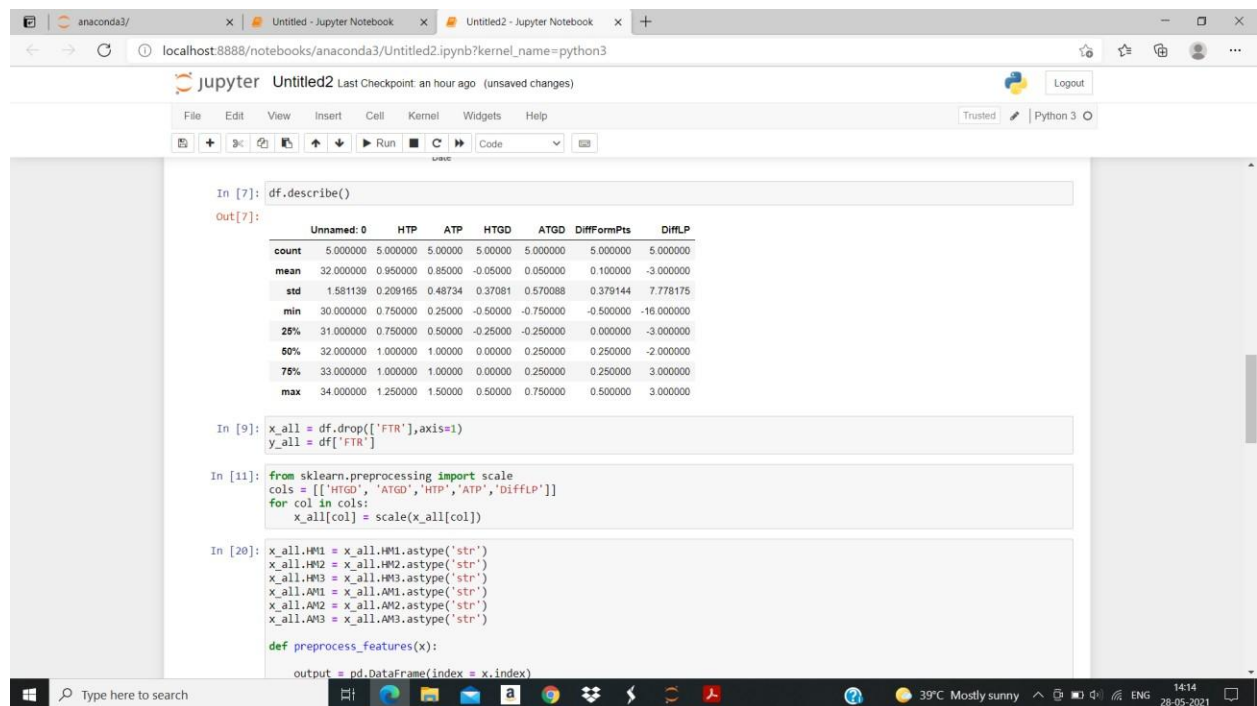
```
In [4]: df.head()
Out[4]:
```

	Unnamed: 0	FTR	HTP	ATP	HM1	HM2	HM3	AM1	AM2	AM3	HTGD	ATGD	DiffFormPts	DiffLP
0	30	H	1.25	1.00	D	D	W	D	W	L	0.50	0.25	0.25	-16
1	31	NH	0.75	0.25	L	L	W	D	L	L	-0.50	-0.75	0.50	-2
2	32	H	1.00	1.00	L	D	W	D	W	L	0.00	0.25	0.00	-3
3	33	NH	0.75	0.50	L	L	W	D	L	D	-0.25	-0.25	0.25	3

Data exploration:



Data Normalization:



The screenshot shows a Jupyter Notebook with the following code and output:

```
In [7]: df.describe()
```

Out[7]:

	Unnamed: 0	HTP	ATP	HTGD	ATGD	DiffFormPts	DiffLP
count	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000
mean	32.000000	0.950000	0.850000	-0.050000	0.050000	0.100000	-3.000000
std	1.581139	0.209165	0.48734	0.37081	0.570088	0.379144	7.778175
min	30.000000	0.750000	0.250000	-0.500000	-0.750000	-0.500000	-16.000000
25%	31.000000	0.750000	0.500000	-0.250000	-0.250000	0.000000	-3.000000
50%	32.000000	1.000000	1.000000	0.000000	0.250000	0.250000	-2.000000
75%	33.000000	1.000000	1.000000	0.000000	0.250000	0.250000	3.000000
max	34.000000	1.250000	1.500000	0.500000	0.750000	0.500000	3.000000

```
In [9]: x_all = df.drop(['FTR'],axis=1)
y_all = df['FTR']

In [11]: from sklearn.preprocessing import scale
cols = [['HTGD', 'ATGD', 'HTP', 'ATP', 'DiffLP']]
for col in cols:
    x_all[col] = scale(x_all[col])

In [20]: x_all.H01 = x_all.H01.astype('str')
x_all.H02 = x_all.H02.astype('str')
x_all.H03 = x_all.H03.astype('str')
x_all.A01 = x_all.A01.astype('str')
x_all.A02 = x_all.A02.astype('str')
x_all.A03 = x_all.A03.astype('str')

def preprocess_features(x):
    output = pd.DataFrame(index = x.index)
```

The bottom of the screenshot shows a Windows taskbar with the date 28-05-2021 and time 14:14.

Code:

import pandas as pd

import numpy as np

import statistics import

seaborn as sns

import matplotlib.pyplot as plt

import math

main = pd.read_csv("C:\\Users\\Thilakraj\\Downloads\\E0.csv")df =

pd.read_csv("C:\\Users\\Thilakraj\\Downloads\\final.csv") df.head()

sns.countplot(x="FTR", data=main)

df.describe()

X_all = df.drop(['FTR'],axis=1)

y_all = df['FTR']


```

from sklearn.preprocessing import scale

cols = [['HTGD','ATGD','HTP','ATP','DiffLP']]

for col in cols:
    X_all[col] = scale(X_all[col])
X_all.HM1 = X_all.HM1.astype('str')
X_all.HM2 = X_all.HM2.astype('str')
X_all.HM3 = X_all.HM3.astype('str')X_all.AM1 = X_all.AM1.astype('str')
X_all.AM2 = X_all.AM2.astype('str')
X_all.AM3 = X_all.AM3.astype('str')
def preprocess_features(X):

    output = pd.DataFrame(index = X.index)
    for col, col_data in X.iteritems():
        if col_data.dtype == object:
            col_data = pd.get_dummies(col_data, prefix = col)
        output = output.join(col_data)
    return output
X_all = preprocess_features(X_all)
print("Processed feature columns ({ } total features):\n{ }".format(len(X_all.columns),
list(X_all.columns)))

```

[illegible]

Importing Packages and csv files:

```
In [1]: import pandas as pd
import numpy as np
import statistics
import seaborn as sns
```

```
In [2]: main = pd.read_csv("C:\\soft computing\\E0.csv")
df = pd.read_csv("C:\\soft computing\\final.csv")
```

```
In [3]: main.head()
```

Out[3]:

	Div	Date	Time	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	...	AvgC<2.5	AHCh	B365CAHH	B365CAHA	PCAHH	PCAHA	MaxCAHI
0	E0	12/09/2020	12:30	Fulham	Arsenal	0	3	A	0	1	...	1.84	0.75	2.01	1.89	2.02	1.91	2.1
1	E0	12/09/2020	15:00	Crystal Palace	Southampton	1	0	H	1	0	...	1.70	0.25	1.78	2.13	1.79	2.17	1.8
2	E0	12/09/2020	17:30	Liverpool	Leeds	4	3	H	3	2	...	2.62	-1.50	1.85	2.05	1.85	2.08	1.9
3	E0	12/09/2020	20:00	West Ham	Newcastle	0	2	A	0	0	...	1.92	-0.50	2.03	1.87	2.04	1.88	2.0
4	E0	13/09/2020	14:00	West Brom	Leicester	0	3	A	0	0	...	1.73	0.25	1.92	1.98	1.93	1.99	1.9

5 rows × 106 columns



```
In [4]: df.head()
```

Out[4]:

	Unnamed: 0	FTR	HTP	ATP	HM1	HM2	HM3	AM1	AM2	AM3	HTGD	ATGD	DiffFormPts	DiffLP
0	30	H	1.25	1.00	D	D	W	D	W	L	0.50	0.25	0.25	-16
1	31	NH	0.75	0.25	L	L	W	D	L	L	-0.50	-0.75	0.50	-2
2	32	H	1.00	1.00	L	D	W	D	W	L	0.00	0.25	0.00	-3
3	33	NH	0.75	0.50	L	L	W	D	L	D	-0.25	-0.25	0.25	3
4	34	NH	1.00	1.50	D	L	W	W	W	L	0.00	0.75	-0.50	3

Data Exploration:

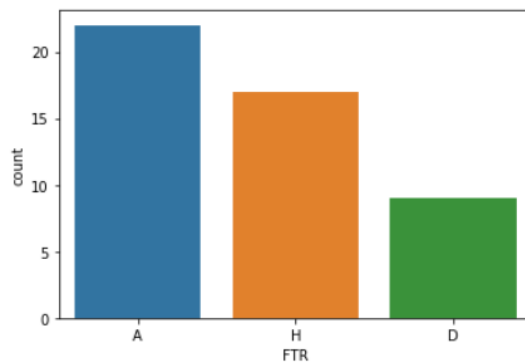
```
In [5]: n_matches = main.shape[0]
n_features = main.shape[1] - 1
n_homewins = len(main[main.FTR == 'H'])
win_rate = (float(n_homewins) / (n_matches)) * 100

print("Total number of matches: {}".format(n_matches))
print("Number of features: {}".format(n_features))
print("Number of matches won by home team: {}".format(n_homewins))
print("Home team win rate: {:.2f}%".format(win_rate))
print("Home team not win rate: {:.2f}%".format(100-win_rate))

sns.countplot(x="FTR", data=main)
```

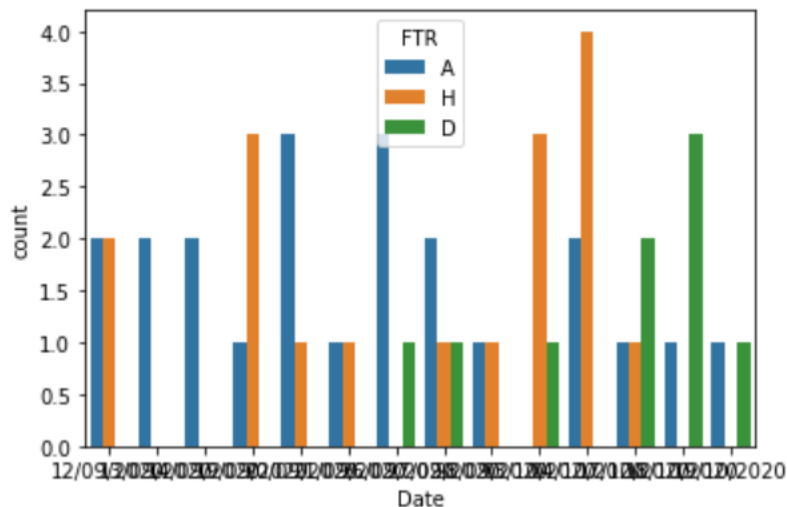
```
Total number of matches: 48
Number of features: 105
Number of matches won by home team: 17
Home team win rate: 35.42%
Home team not win rate: 64.58%
```

```
Out[5]: <AxesSubplot:xlabel='FTR', ylabel='count'>
```



```
In [6]: sns.countplot(x="Date", hue="FTR", data=main)
```

```
Out[6]: <AxesSubplot:xlabel='Date', ylabel='count'>
```



Data Normalization:

```
In [7]: df.describe()
```

```
Out[7]:
```

	Unnamed: 0	HTP	ATP	HTGD	ATGD	DiffFormPts	DiffLP
count	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000
mean	32.000000	0.950000	0.850000	-0.050000	0.050000	0.100000	-3.000000
std	1.581139	0.209165	0.48734	0.37081	0.570088	0.379144	7.778175
min	30.000000	0.750000	0.250000	-0.500000	-0.750000	-0.500000	-16.000000
25%	31.000000	0.750000	0.500000	-0.250000	-0.250000	0.000000	-3.000000
50%	32.000000	1.000000	1.000000	0.000000	0.250000	0.250000	-2.000000
75%	33.000000	1.000000	1.000000	0.000000	0.250000	0.250000	3.000000
max	34.000000	1.250000	1.500000	0.500000	0.750000	0.500000	3.000000

```
In [8]: X_all = df.drop(['FTR'],axis=1)
y_all = df['FTR']
```

```
In [9]: from sklearn.preprocessing import scale
cols = [['HTGD','ATGD','HTP','ATP','DiffLP']]
for col in cols:
    X_all[col] = scale(X_all[col])
```

```
In [10]: X_all.HM1 = X_all.HM1.astype('str')
X_all.HM2 = X_all.HM2.astype('str')
X_all.HM3 = X_all.HM3.astype('str')
X_all.AM1 = X_all.AM1.astype('str')
X_all.AM2 = X_all.AM2.astype('str')
X_all.AM3 = X_all.AM3.astype('str')

def preprocess_features(X):
    output = pd.DataFrame(index = X.index)

    for col, col_data in X.iteritems():

        if col_data.dtype == object:
            col_data = pd.get_dummies(col_data, prefix = col)

        output = output.join(col_data)

    return output

X_all = preprocess_features(X_all)
print ("Processed feature columns ({} total features):\n{}".format(len(X_all.columns), list(X_all.columns)))

Processed feature columns (18 total features):
['Unnamed: 0', 'HTP', 'ATP', 'HM1_D', 'HM1_L', 'HM2_D', 'HM2_L', 'HM3_W', 'AM1_D', 'AM1_W', 'AM2_L', 'AM2_W', 'AM3_D', 'AM3_L',
'HTGD', 'ATGD', 'DiffFormPts', 'DiffLP']
```

```
In [11]: print ("\nFeature values:")
display(X_all.head())
```

Feature values:

	Unnamed: 0	HTP	ATP	HM1_D	HM1_L	HM2_D	HM2_L	HM3_W	AM1_D	AM1_W	AM2_L	AM2_W	AM3_D	AM3_L	HTGD	ATGD	DiffForm
0	30	1.603567	0.344124	1	0	1	0	1	1	0	0	1	0	1	1.658312	0.392232	0
1	31	-1.069045	-1.376494	0	1	0	1	1	1	0	1	0	0	1	-1.356801	-1.568929	0
2	32	0.267261	0.344124	0	1	1	0	1	1	0	0	1	0	1	0.150756	0.392232	0
3	33	-1.069045	-0.802955	0	1	0	1	1	1	0	1	0	1	0	-0.603023	-0.588348	0
4	34	0.267261	1.491202	1	0	0	1	1	0	1	0	1	0	1	0.150756	1.372813	-4

Splitting the Data into Training and Testing:

```
In [14]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=0.5, random_state=2)
```

```
In [15]: from time import time
y_preds = []
def train_classifier(clf, X_train, y_train):

    # Start the clock, train the classifier, then stop the clock
    start = time()
    clf.fit(X_train, y_train)
    end = time()

def predict_labels(clf, features, target):

    start = time()
    y_pred = clf.predict(features)
    y_preds.append(y_pred)
    end = time()

    # Print and return results

    return sum(target == y_pred) / float(len(y_pred))

def train_predict(clf, X_train, y_train, X_test, y_test):

    print("Training a {} using a training set size of {}. . .".format(clf.__class__.__name__, len(X_train)))

    train_classifier(clf, X_train, y_train)

    acc = predict_labels(clf, X_train, y_train)
    print("Accuracy score for training set: ", acc*100, "%")

    acc = predict_labels(clf, X_test, y_test)
    print("Accuracy score for test set: ", acc*100, "%")
```

Applying Three classifiers Model:

```
In [14]: from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

mod_A = LogisticRegression(random_state = 30)
mod_B = MLPClassifier(max_iter= 800,activation='relu',hidden_layer_sizes= 1000)
mod_c = SVC(random_state = 912)

train_predict(mod_A, X_train, y_train, X_test, y_test)
print('')
train_predict(mod_B, X_train, y_train, X_test, y_test)
print('')
train_predict(mod_c, X_train, y_train, X_test, y_test)
print('')
```

Training a LogisticRegression using a training set size of 2. . .
Accuracy score for training set: 100.0 %
Accuracy score for test set: 66.66666666666666 %

Training a MLPClassifier using a training set size of 2. . .
Accuracy score for training set: 100.0 %
Accuracy score for test set: 66.66666666666666 %

Training a SVC using a training set size of 2. . .
Accuracy score for training set: 100.0 %
Accuracy score for test set: 66.66666666666666 %

```
In [15]: print(y_test)
print(y_preds[1])
predss = y_preds[1]

2      H
4     NH
1     NH
Name: FTR, dtype: object
['NH' 'NH' 'NH']
```

```
In [16]: mod_c.predict(X_test)
```

```
Out[16]: array(['NH', 'NH', 'NH'], dtype=object)
```

Output of the Predicted Model:

```
In [17]: print('ACTUAL OUTPUT')
print(' ')
for i in y_test:
    if i == 'H':
        print('Home WIN')
    elif i == 'NH':
        print('Away Team win')
    else:
        print('Draw')
print(' ')
print('PREDICTED OUTPUT')
print(' ')
for i in predss:
    if i == 'H':
        print('Home WIN')
    elif i == 'NH':
        print('Away Team win')
    else:
        print('Draw')
```

ACTUAL OUTPUT

Home WIN
Away Team win
Away Team win

PREDICTED OUTPUT

Away Team win
Away Team win
Away Team win

Source code:

```
import pandas as pd
import numpy as np
import statistics
import seaborn as sns

main = pd.read_csv("C:\\soft computing\\E0.csv")
df = pd.read_csv("C:\\soft computing\\final.csv")

main.head()

df.head()

n_matches = main.shape[0]
n_features = main.shape[1] - 1
n_homewins = len(main[main.FTR == 'H'])
win_rate = (float(n_homewins) / (n_matches)) * 100

print("Total number of matches: {}".format(n_matches))
print("Number of features: {}".format(n_features))
print("Number of matches won by home team: {}".format(n_homewins))
print("Home team win rate: {:.2f}%".format(win_rate))
print("Home team not win rate: {:.2f}%".format(100-win_rate))

sns.countplot(x="FTR", data=main)

sns.countplot(x="Date", hue="FTR", data=main)

df.describe()

X_all = df.drop(['FTR'],axis=1)
```



```

y_all = df['FTR']

from sklearn.preprocessing import scale
cols = [['HTGD','ATGD','HTP','ATP','DiffLP']]
for col in cols:
    X_all[col] = scale(X_all[col])

X_all.HM1 = X_all.HM1.astype('str')
X_all.HM2 = X_all.HM2.astype('str')
X_all.HM3 = X_all.HM3.astype('str')
X_all.AM1 = X_all.AM1.astype('str')
X_all.AM2 = X_all.AM2.astype('str')
X_all.AM3 = X_all.AM3.astype('str')
def preprocess_features(X):
    output = pd.DataFrame(index = X.index)
    for col, col_data in X.iteritems():
        if col_data.dtype == object:
            col_data = pd.get_dummies(col_data, prefix = col)
        output = output.join(col_data)
    return output
X_all = preprocess_features(X_all)

print ("Processed feature columns ({} total features):\n{}".format(len(X_all.columns),
list(X_all.columns)))

print ("\nFeature values:")
display(X_all.head())

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=0.5, random_state=2)

from time import time
y_preds = []

```

```

def train_classifier(clf, X_train, y_train):
    # Start the clock, train the classifier, then stop the clock
    start = time()
    clf.fit(X_train, y_train)
    end = time()

def predict_labels(clf, features, target):
    start = time()
    y_pred = clf.predict(features)
    y_preds.append(y_pred)
    end = time()

# Print and return results
    return sum(target == y_pred) / float(len(y_pred))

def train_predict(clf, X_train, y_train, X_test, y_test):
    print("Training a {} using a training set size of {}".format(clf.__class__.__name__,
len(X_train)))

    train_classifier(clf, X_train, y_train)
    acc = predict_labels(clf, X_train, y_train)
    print("Accuracy score for training set: ", acc*100, "%")
    acc = predict_labels(clf, X_test, y_test)
    print("Accuracy score for test set: ", acc*100, "%")


from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
mod_A = LogisticRegression(random_state = 30)
mod_B = MLPClassifier(max_iter= 800,activation='relu',hidden_layer_sizes= 1000)
mod_c = SVC(random_state = 912)
train_predict(mod_A, X_train, y_train, X_test, y_test)
print("")
train_predict(mod_B, X_train, y_train, X_test, y_test)
print("")
train_predict(mod_c, X_train, y_train, X_test, y_test)

```

```
print("")
```

```
print(y_test)
```

```
print(y_preds[1])
```

```
predss = y_preds[1]
```

```
mod_c.predict(X_test)
```

```
print('ACTUAL OUTPUT')
```

```
print(' ')
```

```
for i in y_test:
```

```
    if i == 'H':
```

```
        print('Home WIN')
```

```
    elif i == 'NH':
```

```
        print('Away Team win')
```

```
    else:
```

```
        print('Draw')
```

```
print(' ')
```

```
print('PREDICTED OUTPUT')
```

```
print(' ')
```

```
for i in predss:
```

```
    if i == 'H':
```

```
        print('Home WIN')
```

```
    elif i == 'NH':
```

```
        print('Away Team win')
```

```
    else:
```

```
        print('Draw')
```

Conclusion:

In this project we have successfully implemented 3 classifier models of neural network to almost(nearly 70%) accurately predict the results of the matches played in England Premiere League. Another feature of the system of prediction is that the later the match predicted, better the accuracy. This clearly highlights the fact that this method of prediction is a learning based method of prediction and that the system predictions improve with more test cases. The error in the system is produced due to the limited data that is used.

References:

- Byungho Min, Jinhyuck Kim, ChongyounChoe and Robert Ian, "A compound framework for sports prediction: The case study of football", Knowledge-Based Systems.
- Y. Y. Petrunin, "Analysis of the football performance: from classical methods to neural network".
- Siraj YouTube videos and GitHub code.
- Dataset Taken link - <http://football-data.co.uk/data.php>