EE514/CS353: Machine Learning Project Report

Session: Spring 2021

Group# 16

Iyad Durrani	2021-02-0111
Umar Razzaq	2021-02-0572
Kash Kumar	2022-11-0104

Department of Electrical Engineering

Syed Babar Ali School of Science and Engineering

Lahore University of Management Sciences

Pakistan

Abstract

This paper aims to predict the outcome of a given match in the English Premier League (EPL) using multiple classification algorithms and analyze and report the differences between the performance of the classifiers. There are 20 CSV files with the rows being a particular match, one for each year of the EPL from 2000-2019. For each match, we have a few characteristics about the match, some of them are: which team was the home team, what was the number of shots by each team, what was the number of shots on target, what were the betting statistics of the match, and many more.We also have the final standings of the 43 teams in each year consolidated into a single file, each row represents a team and each attribute is the year of the standing. Using the given data, in this paper, we will apply feature engineering after cleaning and preprocessing the data, and implement the following classifiers: KNN, Logistic Regression, Bayes Classifier, SVM, and a Neural Network, and then finally report on the differences in performance of each classifier. We concluded that our classifiers perform best with feature normalisation and using PCA to reduce the dimensionality of data. All classifiers except logistic regression were giving us an accuracy of about 60% and the reason why logistic regression may not be performing as well was due to outliers in the data.

ANALYSIS OF ENGLISH PREMIER LEAGUE USING MACHINE LEARNING ALGORITHMS

1.1 Introduction

The objective of this report is to predict the results of English Premier League (EPL), one of the most-watched sports leagues around the world. By using various Machine Learning algorithms we learned from the historical data to predict the results of EPL.

The origin of Machine Learning since 1952 has revolutionized the concept of traditional programming by using computational methods and automating the processes through iterations and creating self-learning algorithms to solve complex mathematical calculations to speed user insights and minimize decision bias. By incorporating cross-domain integrations in surveillance, neuroscience, healthcare, security, and other various other fields, Machine Learning has become an integral part of our lives.

One such aspect of Machine Learning used in this report to build applications that exhibit iterative improvement is Supervised Learning. It will initially learn from the given training data to assess the performance of each team over others until it is ready to accurately test and approximate the nature of the relationship between given team results to predict their respective outputs.

To achieve the accurate prediction on EPL matches results with final team standings on the table, we categorized the dataset into a set of different classes for a given data point by approximating the mapping function from the given sample to discrete output variables into binary or multi-classes using various learning algorithms including but not limited to KNN, Logistic Regression, SVM and Neural Networks.

We incorporated our machine learning models over 19 years of historical data to design a well-generalized system for predictive modeling. Considering prior researchers suggested that the home/away factor is an important characteristic of the problem, in our analysis, we computed the feature values for both the home and away teams for every feature that we engineered. Hence, this paper also highlights the mathematical formulations of Features Engineering as it was the crucial aspect of this research in predicting the given match outcome. Moreover, since many features increase our space complexity, we selected the best performing features after performing significant features engineering.

1.1.1 Classification Algorithms Used in the Model:

1. **Naïve Bayes Classifier (Generative classifier):** Purely probabilistic method which uses the principles of conditional probability to compute the posterior probabilities and assumes that features are independent/ unrelated to other features' presence.

2. **Neural Networks:** Algorithm series' that attempts to identify the relationship in a set of data through a process that mimics the way the human brain operates.

3. **K-Nearest Neighbors (Instance-based classifier):** works under the assumption that closest points must be similar and hence classifies them with the closest group cluster.

4. **SVM:** Relatively more efficient and highly effective in high dimensional spaces as it uses only a subset of the training set. However, it does not directly provide the probability estimates.

5. **Logistic Regression (Discriminative classifier):** Finds the best-fitting relationship between dependent and independent variables by explaining how the dependent variable is affected by a set of independent variables and assumes independence among predictors.

1.1.2 Performance Evaluation Metrics: After fitting models with each of the algorithms mentioned above, we used the following metrics to evaluate the performance of each of the algorithms.

1) **Model's Accuracy:** It determines the correct % of predictions made by the model. However, it might not be a good metric for imbalanced datasets.

2) **Model's Sensitivity/Recall:** It is the fraction of relevant instances that are retrieved during pattern recognition.

3) **Model's Specificity/Precision:** Since it only identifies the proportion of the relevant data points, the goodness of precision depends on the model's objective and data type.
4) **The F1 Score:** It is the harmonic mean of a model's Sensitivity and Specificity, it tells us how precise our model is and how robust it is.

1.2 Mathematical Formulation

$D = \{(x1, y1), (x2, y2), ..., (xn, yn)\} \subseteq Xd * y$

where,

$y = \{homewin, homeloose, draw\}$(3 class classification)

$X = \{'B365H', 'B365D', 'B365A', ..., 'Away\ Team\ Standing\ In\ Last\ Season'\}$

(Features are explained in the feature engineering section)

W are the weights of each feature (Relative)

1.2.1 KNN:

For a match x $\in X^d$, we define a set of $S_X \subseteq D$ as a set of k neighbors. Using the following function dist that computes the distance between two matches in X^d, S_X of size k as

$dist(x, x') \geq max\ dist(x, x''),\ (x', y')\ \forall \in D \setminus S_X$

Following distances can be used in KNN:

- Minkowski: dist(x,x') = $||x - x'||_p = (\sum_{i=1}^{d}(|x_i - x'_i|)^p)^{1/p}$, here if
  - p = 1, we have Manhattan distance $\sum_{i=1}^{d}|x_i - x'_i|$ and $||x - x'||_1$ =
  - p = 2, we have Euclidean distance and $||x - x'||_2 = \sqrt{\Sigma_{i=1}^{d}(xi - xi')\hat{}2}$
  - p = ∞, we have Chebyshev distance and $||x - x'||_\infty = \max_{i=1,2,\ldots,d}(|x_i - x'_i|)$
  - where distance is a scalar; x, x' are two points in n-space; and xi, xi' are feature vectors starting from the origin of the space is the feature vector explained above
- Cosine Distance: $\mathbf{x}^T\mathbf{x}' = \|\mathbf{x}\|_2\|\mathbf{x}'\|_2 \cos\theta$

1.2.2 Naive Bayes Classifier:

The hypothesis class is denoted by H. for our given problem, we wish to select the best hypothesis class h⊆H.

Let, h = {homeTeamWins, homeTeamDraws, homeTeamLosses}

D = featuresOfHomeAndAwayTeam

Find h that maximises P(h|D)

P(h|D) = (P(D|h) * P(h)) / (P(D))

h_{MAP} = maximize P(h|D) = maximize_h (P(D|h) * P(h)) / (P(D)) => h_{ML} = maximize_h P(D|h)

1.2.3 Logistic Regression:

The logistic regression model predicts for each y class (home team wins, losses or draws) as follows:

$$P(y=1|x) = h_\Theta(x) = \sigma(\Theta_0 + \Theta_1 x) = \frac{1}{(1+e-(\Theta 0 + \Theta 1x))}$$

Here y is the outcome of the match in terms of the home team and x is the feature vector.

It standardizes the output from given in the range of infinite values in the form of probability between 0 to 1 with the help of Sigmoid function σ. Where Θ_0 and Θ_1 bias term and model parameters respectively.

1.2.4 Soft SVM

Minimize w, θ $(||w||^2 + C(\sum_{i=1}^{n} \epsilon i) = =(w^T w + C(\sum_{i=1}^{n} \epsilon i)$

subject to: $y_i (w^T X_i - \theta) \geq (1 - \epsilon i)$, $\epsilon i \geq 0$

Where ϵi is the slack variable trying to optimize the problem and loss and parameter C maintains the relative trade-off between the importance of maximizing the margin and fitting the training data

1.2.5 Neural Network

Pre-Activation Aggregation: $z = w^T X + b1$

- Here X is the feature vector (it contains the information for each match)
- Where z is (1x n), w^T is (1xd), X is (dxn) and b1 is (1xn) matrices

For Sigmoid Activation Function, we get: $a = g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$ where a is a row vector of length n, and ith entry represents an output for ith output, i.e. either 0 or 1.

1.3 Data-Preprocessing

After importing the dataset from OneDrive for Business (sharepoint.com) for 20 seasons of EPL, we imported the following relevant libraries for the utilization in our project analysis: Pandas, Numpy, glob, matplotlib, and other various libraries from sklearn. Using Pandas we loaded the data of each season into their own DataFrames. Apart from the Seasons data, we also had the standings of each time in the seasons.

After examining the dataframes for each season, we chose the data of 2009-2019 (10 Seasons) for training and testing our model as there were no betting stats available for seasons before 2009. So in order to maintain the same features across all our data, we went with the Seasons 2009-2019. After doing Feature Engineering on all Seasons (explained in the next part), we stacked all the Seasons individual DataFrames together into one.

In order to transform the raw data into proper format and enhance its quality to generate meaningful insights, we followed the following steps to achieve this goal:

- Data Cleaning:
    - Missing values: Since the lowest and highest rank of any team in the tournament could be between 1 and 20, we used 21 for any missing values (teams that did not participate in a specific year) to specify to the model that that team did not take part in the preceding season.
- Data Splitting and Transformation:
    - Feature scaling: We used the formula (x-x.mean)/std(x), known as normalisation, to bring all our data into a mutual scale.
    - Splitting the dataset into train and test: After stacking all the seasons matches together, we ended up with 3580 matches in total. Of which we used the initial 3400 matches to train our model and the last 180 matches for predicting and testing our model.
- Data Reduction:
    - Principal Component Analysis: used to reduce attributes by aggregating highly correlated attributes together.

1.4 Feature Engineering

1.4.1 Feature Extraction And Data Preparation

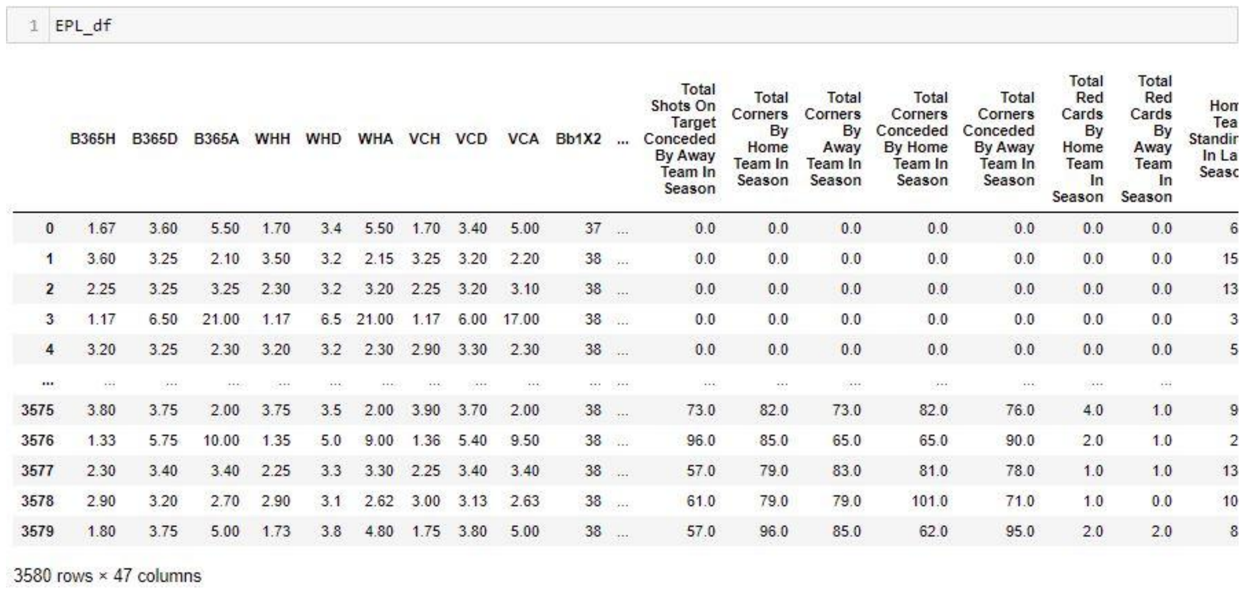
Firstly, we used the full-time home goal and full-time away goals to make a feature that told us whether the home team won, lost, or drew the match. This would be our y values. We used the columns in our 10 seasons data set to do some feature extraction. The Features that we extracted from the raw data are:

1. 'Total Goals By Home Team In Season',
2. 'Total Goals By Away Team In Season',
3. 'Total Goals Conceded By Home Team In Season',
4. 'Total Goals Conceded By Away Team In Season',
5. 'Points Of Home Team In Season',
6. 'Points Of Away Team In Season',
7. 'Points Of Home Team In Last 5 Games Of Season',
8. 'Points Of Away Team In Last 5 Games Of Season',
9. 'Total Shots By Home Team In Season',
10. 'Total Shots By Away Team In Season',
11. 'Total Shots Conceded By Home Team In Season',
12. 'Total Shots Conceded By Away Team In Season',
13. 'Total Shots On Target By Home Team In Season',
14. 'Total Shots On Target By Away Team In Season',
15. 'Total Shots On Target Conceded By Home Team In Season',
16. 'Total Shots On Target Conceded By Away Team In Season',
17. 'Total Corners By Home Team In Season',
18. 'Total Corners By Away Team In Season',
19. 'Total Corners Conceded By Home Team In Season',
20. 'Total Corners Conceded By Away Team In Season',
21. 'Total Red Cards By Home Team In Season',
22. 'Total Red Cards By Away Team In Season',
23. 'Home Team Standing In Last Season',
24. 'Away Team Standing In Last Season'
25. 'B365H' (Bet365 home win odds)
26. 'B365D' (Bet365 draw odds)
27. 'B365A' (Bet365 away win odds)
28. ‘WHH’ (William Hill home win odds)
29. 'WHD' (William Hill draw odds)
30. 'WHA' (William Hill away win odds)
31. 'VCH' (VC Bet home win odds)
32. 'VCD' (VC Bet draw odds)
33. 'VCA’ (VC Bet away win odds)
34. 'Bb1X2' (Number of BetBrain bookmakers used to calculate match odds averages and maximums)
35. 'BbMxH' (Betbrain maximum home win odds)
36. 'BbAvH' (Betbrain average home win odds)

37. 'BbMxD' (Betbrain maximum draw odds)
38. 'BbAvD' (Betbrain average draw win odds)
39. 'BbMxA' (Betbrain maximum away win odds)
40. 'BbAvA' (Betbrain average away win odds)
41. 'BbOU' (Number of BetBrain bookmakers used to calculate over/under 2.5 goals (total goals) averages and maximums)
42. 'BbMx>2.5' (Betbrain maximum over 2.5 goals)
43. 'BbAv>2.5' (Betbrain average over 2.5 goals)
44. 'BbMx<2.5' (Betbrain maximum under 2.5 goals)
45. 'BbAv<2.5' (Betbrain average under 2.5 goals)

These features were calculated to give us an idea of how a particular team has been performing in the past. This helps the model predict which of the two teams will win or lose or if the match will end up being drawn. Once we train our models, we will revisit this feature selection and extraction phase to evaluate if any other features could be added to improve the performance of our classifiers.

After extracting common features in all the years of data we had, we concatenate the data to give us one file. Furthermore, we drop the raw data columns that are not going to be used in our model. This includes all data columns other than the mentioned extracted features above.



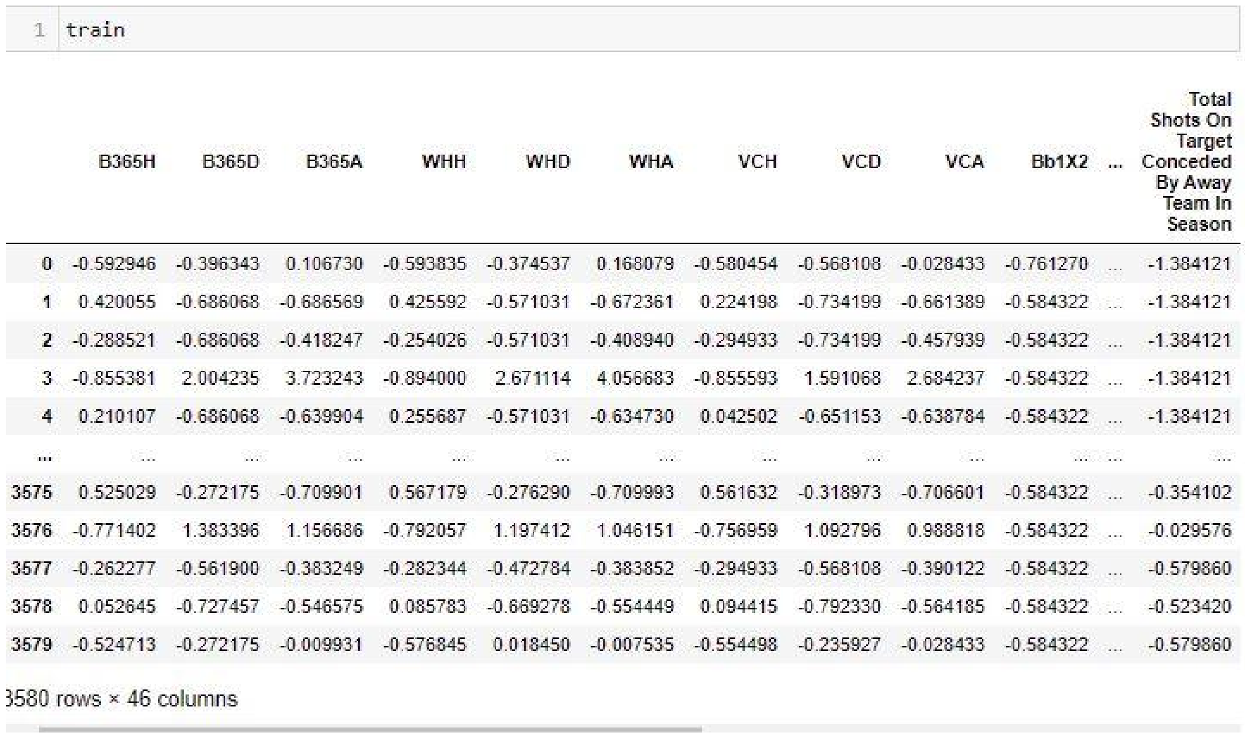
```
EPL_df
```

	B365H	B365D	B365A	WHH	WHD	WHA	VCH	VCD	VCA	Bb1X2	...	Total Shots On Target Conceded By Away Team In Season	Total Corners By Home Team In Season	Total Corners By Away Team In Season	Total Corners Conceded By Home Team In Season	Total Corners Conceded By Away Team In Season	Total Red Cards By Home Team In Season	Total Red Cards By Away Team In Season	Hom Tea Standir In La Seasc
0	1.67	3.60	5.50	1.70	3.4	5.50	1.70	3.40	5.00	37	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6
1	3.60	3.25	2.10	3.50	3.2	2.15	3.25	3.20	2.20	38	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	15
2	2.25	3.25	3.25	2.30	3.2	3.20	2.25	3.20	3.10	38	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	13
3	1.17	6.50	21.00	1.17	6.5	21.00	1.17	6.00	17.00	38	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3
4	3.20	3.25	2.30	3.20	3.2	2.30	2.90	3.30	2.30	38	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3575	3.80	3.75	2.00	3.75	3.5	2.00	3.90	3.70	2.00	38	...	73.0	82.0	73.0	82.0	76.0	4.0	1.0	9
3576	1.33	5.75	10.00	1.35	5.0	9.00	1.36	5.40	9.50	38	...	96.0	85.0	65.0	65.0	90.0	2.0	1.0	2
3577	2.30	3.40	3.40	2.25	3.3	3.30	2.25	3.40	3.40	38	...	57.0	79.0	83.0	81.0	78.0	1.0	1.0	13
3578	2.90	3.20	2.70	2.90	3.1	2.62	3.00	3.13	2.63	38	...	61.0	79.0	79.0	101.0	71.0	1.0	0.0	10
3579	1.80	3.75	5.00	1.73	3.8	4.80	1.75	3.80	5.00	38	...	57.0	96.0	85.0	62.0	95.0	2.0	2.0	8

3580 rows × 47 columns

1.4.2 Scaling and PCA

After extracting the useful and common features for a match, we normalised our features to bring them onto the same scale.



```
train
```

	B365H	B365D	B365A	WHH	WHD	WHA	VCH	VCD	VCA	Bb1X2	...	Total Shots On Target Conceded By Away Team In Season
0	-0.592946	-0.396343	0.106730	-0.593835	-0.374537	0.168079	-0.580454	-0.568108	-0.028433	-0.761270	...	-1.384121
1	0.420055	-0.686068	-0.686569	0.425592	-0.571031	-0.672361	0.224198	-0.734199	-0.661389	-0.584322	...	-1.384121
2	-0.288521	-0.686068	-0.418247	-0.254026	-0.571031	-0.408940	-0.294933	-0.734199	-0.457939	-0.584322	...	-1.384121
3	-0.855381	2.004235	3.723243	-0.894000	2.671114	4.056683	-0.855593	1.591068	2.684237	-0.584322	...	-1.384121
4	0.210107	-0.686068	-0.639904	0.255687	-0.571031	-0.634730	0.042502	-0.651153	-0.638784	-0.584322	...	-1.384121
...	...	...	...	...	...	...	...	...	...	...	...	...
3575	0.525029	-0.272175	-0.709901	0.567179	-0.276290	-0.709993	0.561632	-0.318973	-0.706601	-0.584322	...	-0.354102
3576	-0.771402	1.383396	1.156686	-0.792057	1.197412	1.046151	-0.756959	1.092796	0.988818	-0.584322	...	-0.029576
3577	-0.262277	-0.561900	-0.383249	-0.282344	-0.472784	-0.383852	-0.294933	-0.568108	-0.390122	-0.584322	...	-0.579860
3578	0.052645	-0.727457	-0.546575	0.085783	-0.669278	-0.554449	0.094415	-0.792330	-0.564185	-0.584322	...	-0.523420
3579	-0.524713	-0.272175	-0.009931	-0.576845	0.018450	-0.007535	-0.554498	-0.235927	-0.028433	-0.584322	...	-0.579860

3580 rows × 46 columns

Then we used PCA to reduce the dimensionality of our data, this would also help our models to perform better. We used PCA to retain 95% of the variance of our features, thus helping us greatly reduce the number of features, yet maintaining 95% of the useful information in the features. We ended up with 12 features from the 45 we started from, which reduced our dimensionality greatly.

1.5 Implementation:

Our classification models classified the EPL match results on the basis of each teams historical attributes.

We used the Sklearn library to train our models and report on their accuracies.

Firstly, we trained our classifiers on the features that were not normalised and were not reduced by using PCA. We did this to get a baseline reading of the performance of our models and then see how each step would improve our models accuracies.

Next, we normalised our features and performed PCA on them. This gave us the best accuracies for all models.

Furthermore, in the above steps we used a 95-5 train test split (3400-180 matches). The reason why we used such a high percentage for the training set was because we noticed, as we increased the size of our training set we were getting better accuracies and F1 scores. This meant that we are being limited by the amount of test data we had to train our models effectively.

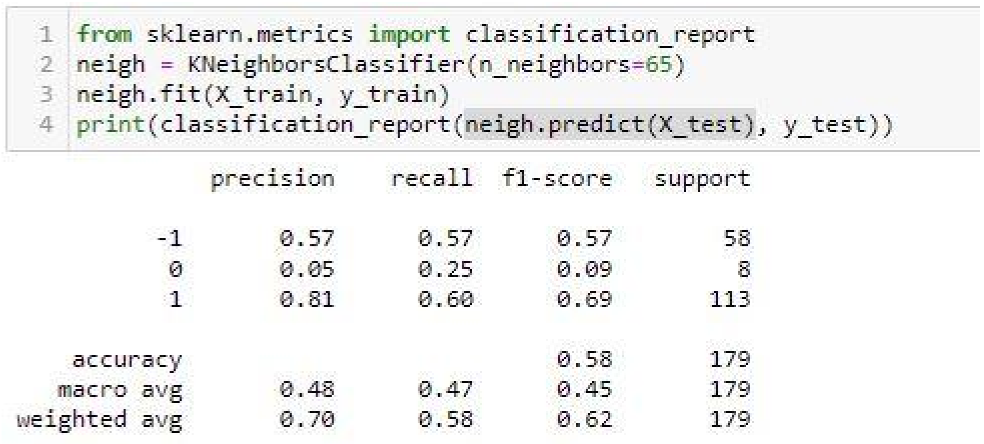
In order to compare, we also ran all our models on a more traditional 80-20 train test split. And the results were as expected, we were getting lower accuracies and F1 scores in this case.

Lastly, our models were giving an extremely low score on the 'draw' F1 scores, upon further investigation, we found out that our training labels were imbalanced. This was adding a bias in our model, which wasnt letting it correctly predict draws. Once we included equal labels for all 3 classes in our train set (removed instances of the class that was larger), our F1 score of 'draw' improved. But it also worsened our overall F1 score and accuracies, which further points towards not having adequate amount of training data.

1.5.1. Models Without Scaling and PCA:

This was done to get a baseline reading of the performance of our models.

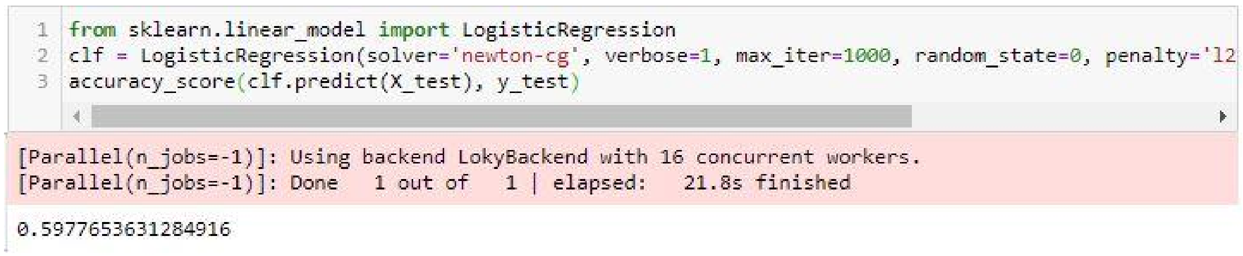
1.5.1.1. K-Nearest Neighbors Classifier:



```python
from sklearn.metrics import classification_report
neigh = KNeighborsClassifier(n_neighbors=65)
neigh.fit(X_train, y_train)
print(classification_report(neigh.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.57      0.57      0.57        58
           0       0.05      0.25      0.09         8
           1       0.81      0.60      0.69       113

    accuracy                           0.58       179
   macro avg       0.48      0.47      0.45       179
weighted avg       0.70      0.58      0.62       179
```

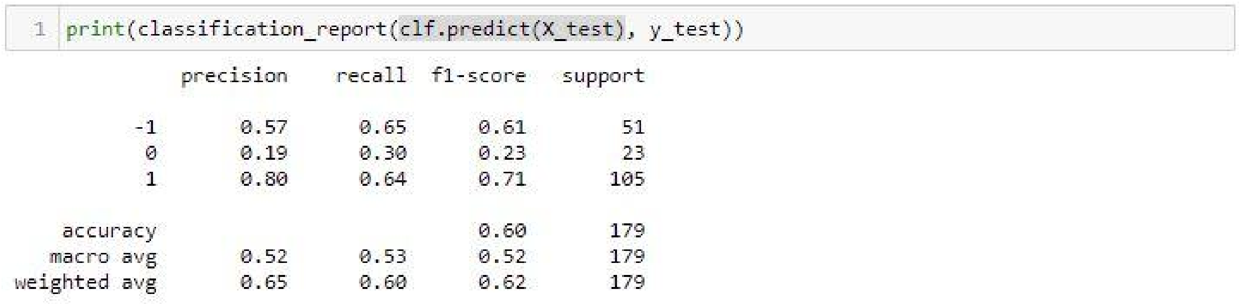
1.5.1.2. Logistic Regression Classifier:



```python
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(solver='newton-cg', verbose=1, max_iter=1000, random_state=0, penalty='l2
accuracy_score(clf.predict(X_test), y_test)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   21.8s finished
```

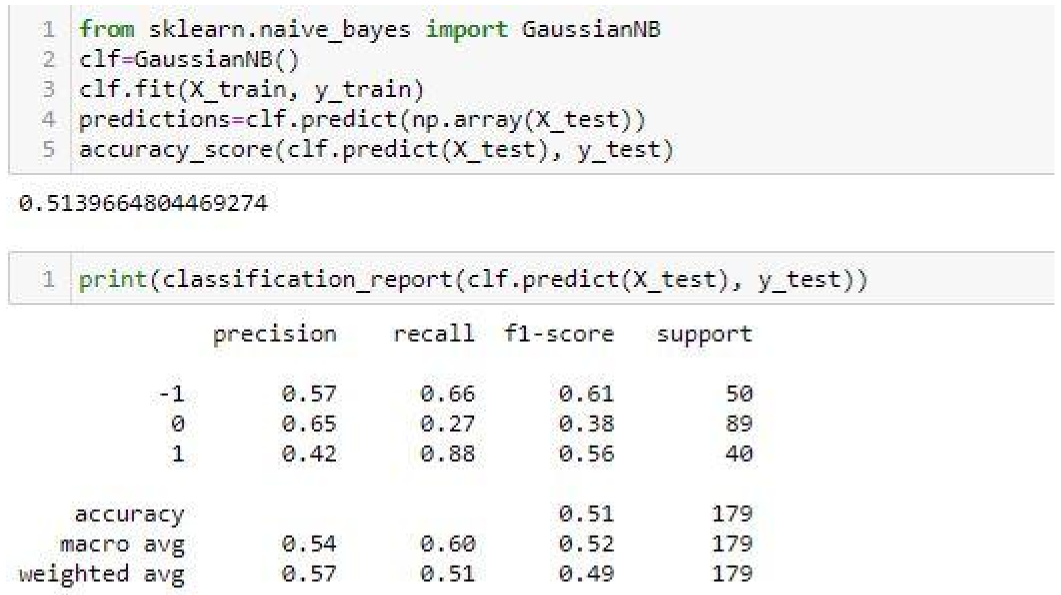
```
0.5977653631284916
```



```python
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.57      0.65      0.61        51
           0       0.19      0.30      0.23        23
           1       0.80      0.64      0.71       105

    accuracy                           0.60       179
   macro avg       0.52      0.53      0.52       179
weighted avg       0.65      0.60      0.62       179
```

1.5.1.3. Naive Bayes Classifier:



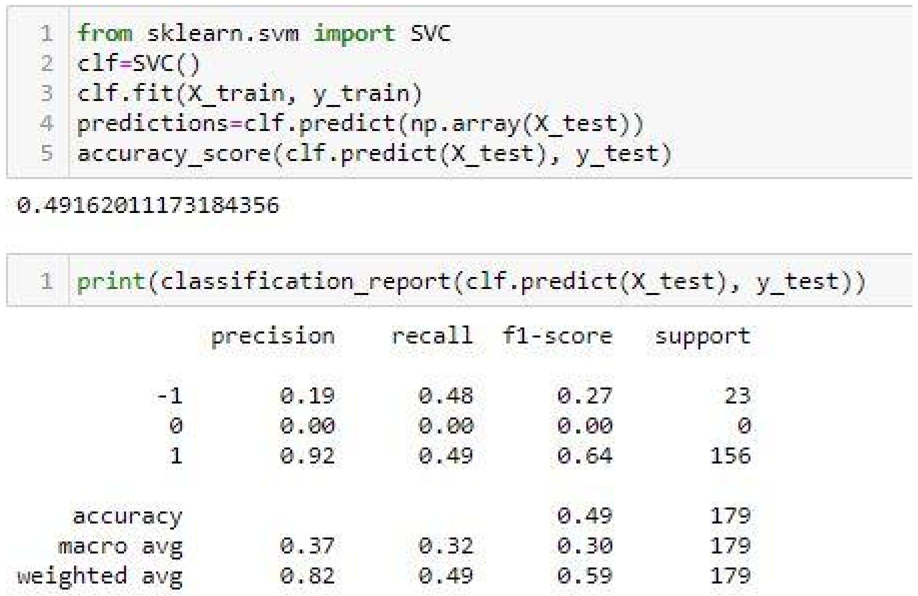
```
from sklearn.naive_bayes import GaussianNB
clf=GaussianNB()
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.5139664804469274
```

```
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.57      0.66      0.61        50
           0       0.65      0.27      0.38        89
           1       0.42      0.88      0.56        40

    accuracy                           0.51       179
   macro avg       0.54      0.60      0.52       179
weighted avg       0.57      0.51      0.49       179
```

1.5.1.4. Support Vector Machine Classifier:



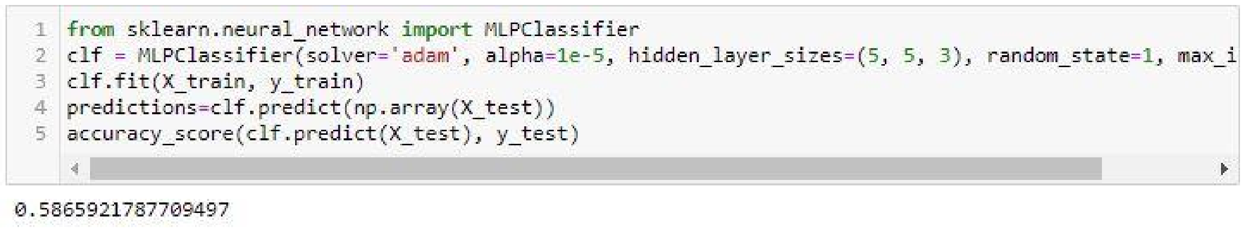
```
from sklearn.svm import SVC
clf=SVC()
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.49162011173184356
```

```
print(classification_report(clf.predict(X_test), y_test))
```

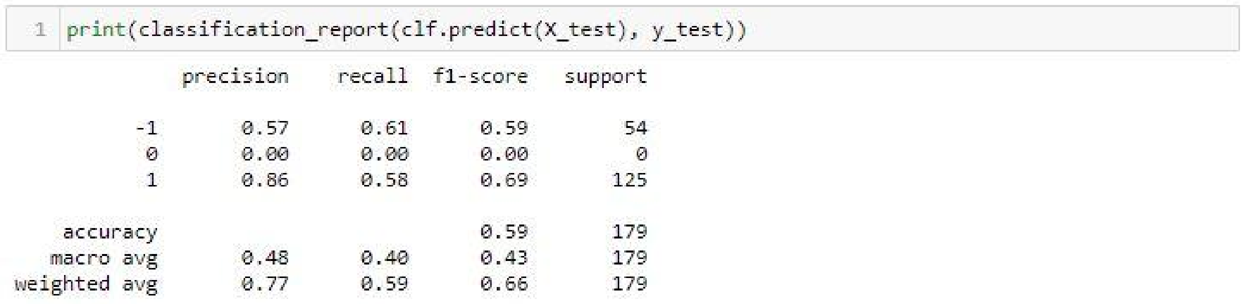
```
              precision    recall  f1-score   support

          -1       0.19      0.48      0.27        23
           0       0.00      0.00      0.00         0
           1       0.92      0.49      0.64       156

    accuracy                           0.49       179
   macro avg       0.37      0.32      0.30       179
weighted avg       0.82      0.49      0.59       179
```

1.5.1.5. Neural Network Classifier:



```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(5, 5, 3), random_state=1, max_i
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.5865921787709497
```

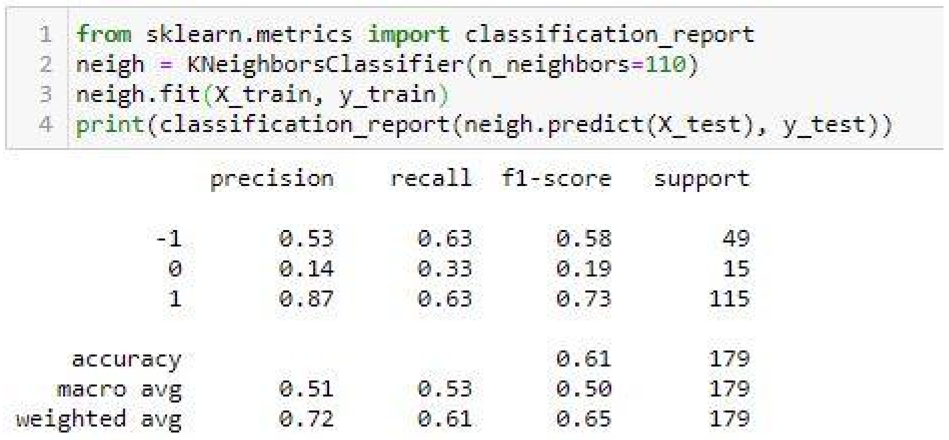


```
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.57      0.61      0.59        54
           0       0.00      0.00      0.00         0
           1       0.86      0.58      0.69       125

    accuracy                           0.59       179
   macro avg       0.48      0.40      0.43       179
weighted avg       0.77      0.59      0.66       179
```

1.5.2 Models With Scaling and PCA:

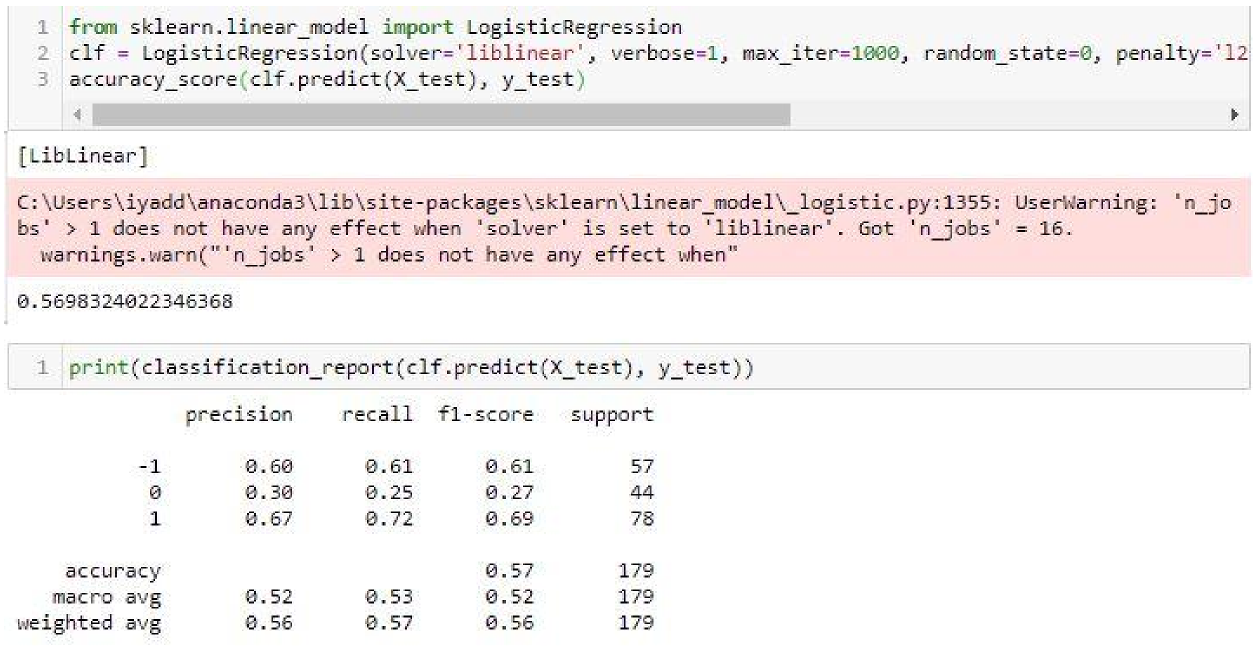
1.5.2.1. K-Nearest Neighbors Classifier:



```
from sklearn.metrics import classification_report
neigh = KNeighborsClassifier(n_neighbors=110)
neigh.fit(X_train, y_train)
print(classification_report(neigh.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.53      0.63      0.58        49
           0       0.14      0.33      0.19        15
           1       0.87      0.63      0.73       115

    accuracy                           0.61       179
   macro avg       0.51      0.53      0.50       179
weighted avg       0.72      0.61      0.65       179
```

1.5.2.2. Logistic Regression Classifier:



```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(solver='liblinear', verbose=1, max_iter=1000, random_state=0, penalty='l2
accuracy_score(clf.predict(X_test), y_test)
```

```
[LibLinear]
```

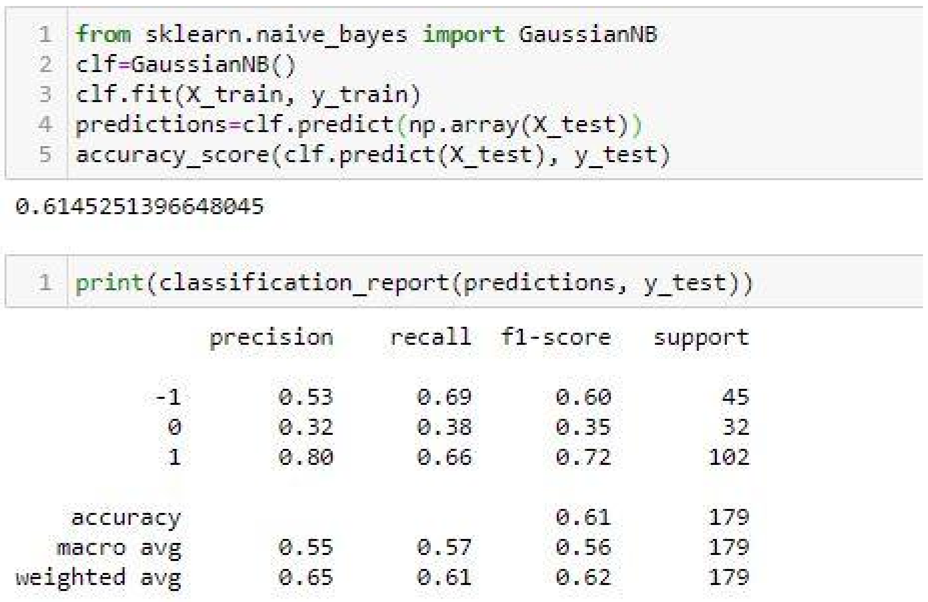
```
C:\Users\iyadd\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1355: UserWarning: 'n_jo
bs' > 1 does not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 16.
  warnings.warn("'n_jobs' > 1 does not have any effect when"
```

```
0.5698324022346368
```

```
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.60      0.61      0.61        57
           0       0.30      0.25      0.27        44
           1       0.67      0.72      0.69        78

    accuracy                           0.57       179
   macro avg       0.52      0.53      0.52       179
weighted avg       0.56      0.57      0.56       179
```

1.5.2.3. Naive Bayes Classifier:



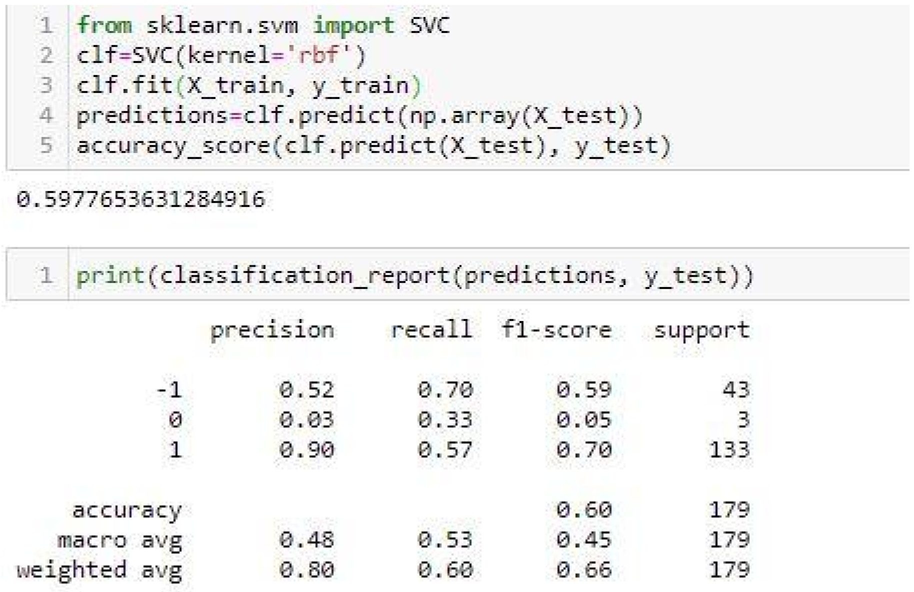
```
from sklearn.naive_bayes import GaussianNB
clf=GaussianNB()
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.6145251396648045
```

```
print(classification_report(predictions, y_test))
```

```
              precision    recall  f1-score   support

          -1       0.53      0.69      0.60        45
           0       0.32      0.38      0.35        32
           1       0.80      0.66      0.72       102

    accuracy                           0.61       179
   macro avg       0.55      0.57      0.56       179
weighted avg       0.65      0.61      0.62       179
```

1.5.2.4. Support Vector Machine Classifier:



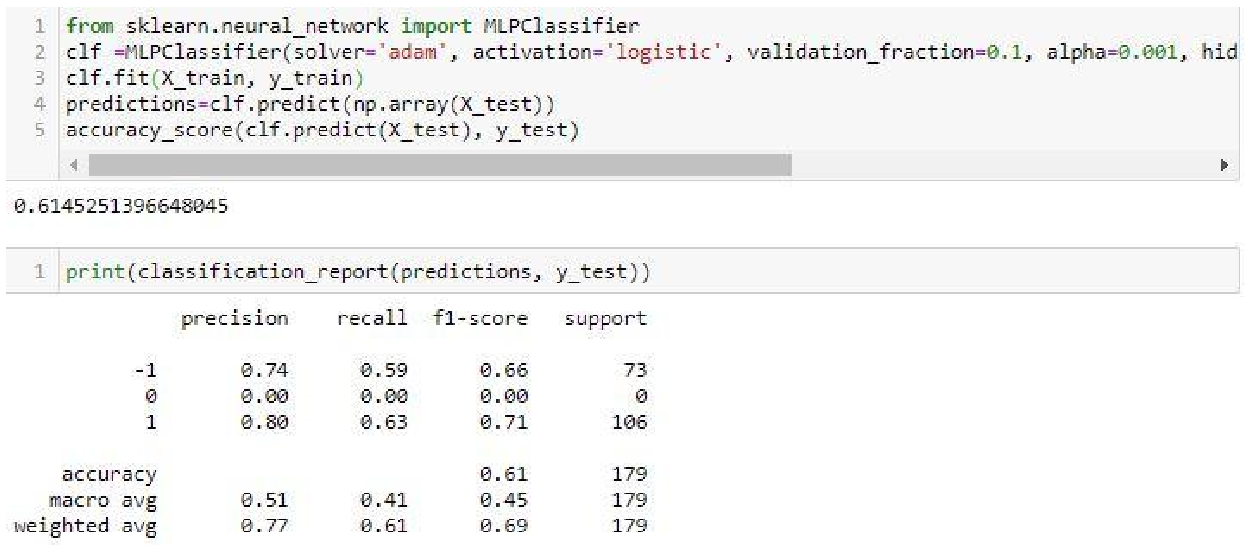
```python
from sklearn.svm import SVC
clf=SVC(kernel='rbf')
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.5977653631284916
```

```python
print(classification_report(predictions, y_test))
```

```
              precision    recall  f1-score   support

          -1       0.52      0.70      0.59        43
           0       0.03      0.33      0.05         3
           1       0.90      0.57      0.70       133

    accuracy                           0.60       179
   macro avg       0.48      0.53      0.45       179
weighted avg       0.80      0.60      0.66       179
```

1.5.2.5. Neural Network Classifier:



```python
from sklearn.neural_network import MLPClassifier
clf =MLPClassifier(solver='adam', activation='logistic', validation_fraction=0.1, alpha=0.001, hid
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

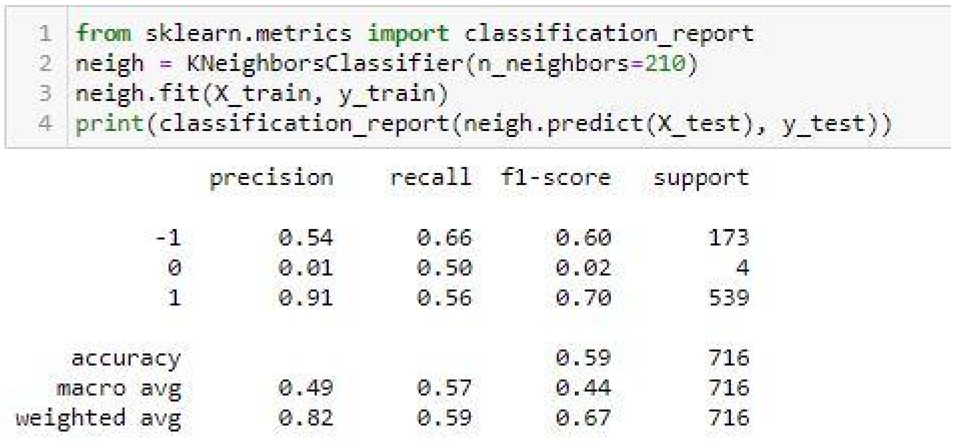
```
0.6145251396648045
```

```python
print(classification_report(predictions, y_test))
```

```
              precision    recall  f1-score   support

          -1       0.74      0.59      0.66        73
           0       0.00      0.00      0.00         0
           1       0.80      0.63      0.71       106

    accuracy                           0.61       179
   macro avg       0.51      0.41      0.45       179
weighted avg       0.77      0.61      0.69       179
```

1.5.3 Models Lower Training Set (80-20 split):

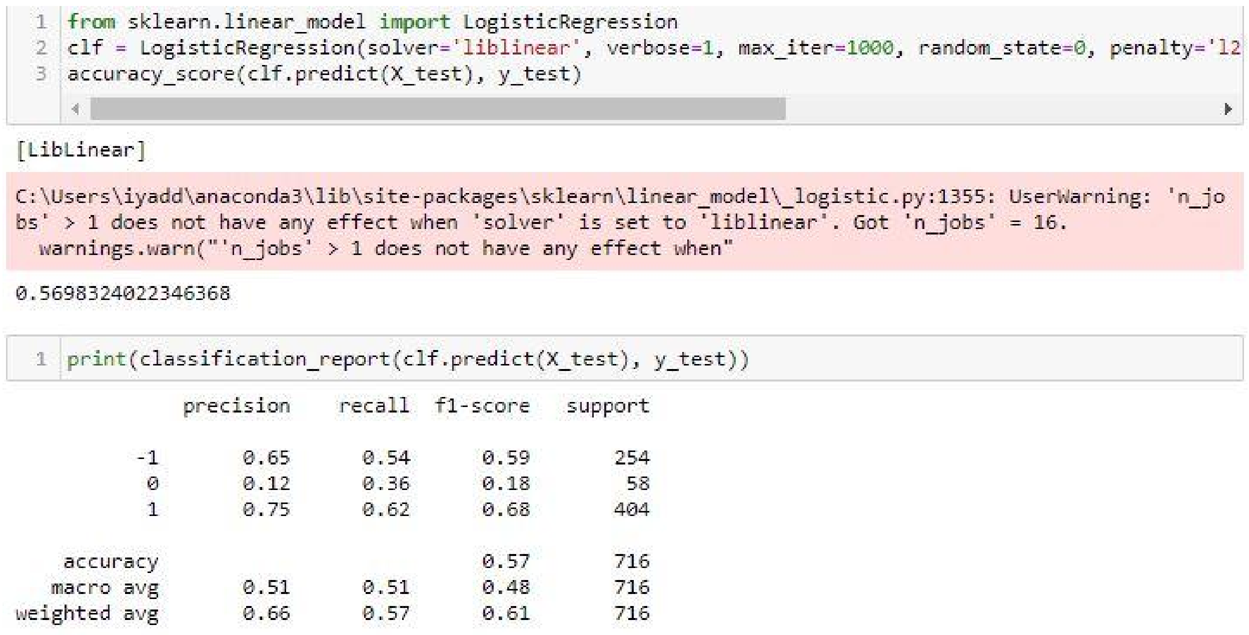
1.5.3.1. K-Nearest Neighbors Classifier:



```
from sklearn.metrics import classification_report
neigh = KNeighborsClassifier(n_neighbors=210)
neigh.fit(X_train, y_train)
print(classification_report(neigh.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.54      0.66      0.60       173
           0       0.01      0.50      0.02         4
           1       0.91      0.56      0.70       539

    accuracy                           0.59       716
   macro avg       0.49      0.57      0.44       716
weighted avg       0.82      0.59      0.67       716
```

1.5.3.2. Logistic Regression Classifier:



```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(solver='liblinear', verbose=1, max_iter=1000, random_state=0, penalty='l2
accuracy_score(clf.predict(X_test), y_test)
```

```
[LibLinear]
```

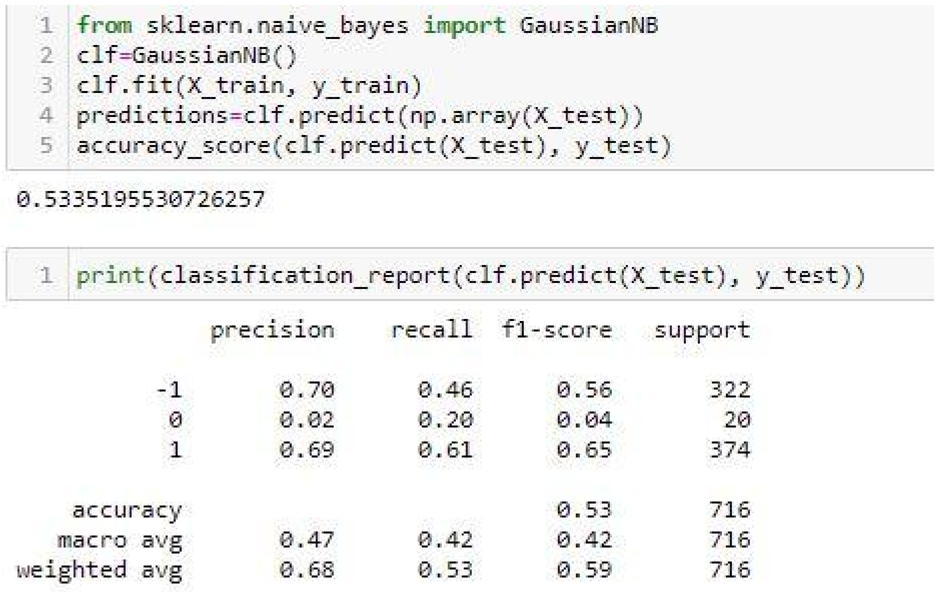
```
C:\Users\iyadd\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1355: UserWarning: 'n_jo
bs' > 1 does not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 16.
  warnings.warn("'n_jobs' > 1 does not have any effect when"
```

```
0.5698324022346368
```

```
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.65      0.54      0.59       254
           0       0.12      0.36      0.18        58
           1       0.75      0.62      0.68       404

    accuracy                           0.57       716
   macro avg       0.51      0.51      0.48       716
weighted avg       0.66      0.57      0.61       716
```

1.5.3.3. Naive Bayes Classifier:



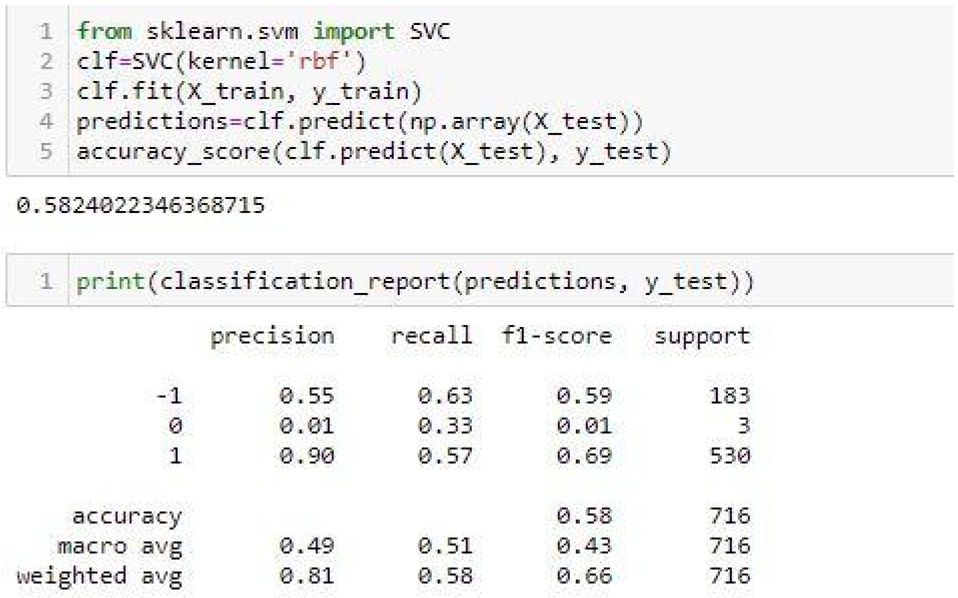
```
from sklearn.naive_bayes import GaussianNB
clf=GaussianNB()
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.5335195530726257
```

```
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.70      0.46      0.56       322
           0       0.02      0.20      0.04        20
           1       0.69      0.61      0.65       374

    accuracy                           0.53       716
   macro avg       0.47      0.42      0.42       716
weighted avg       0.68      0.53      0.59       716
```

1.5.3.4. Support Vector Machine Classifier:



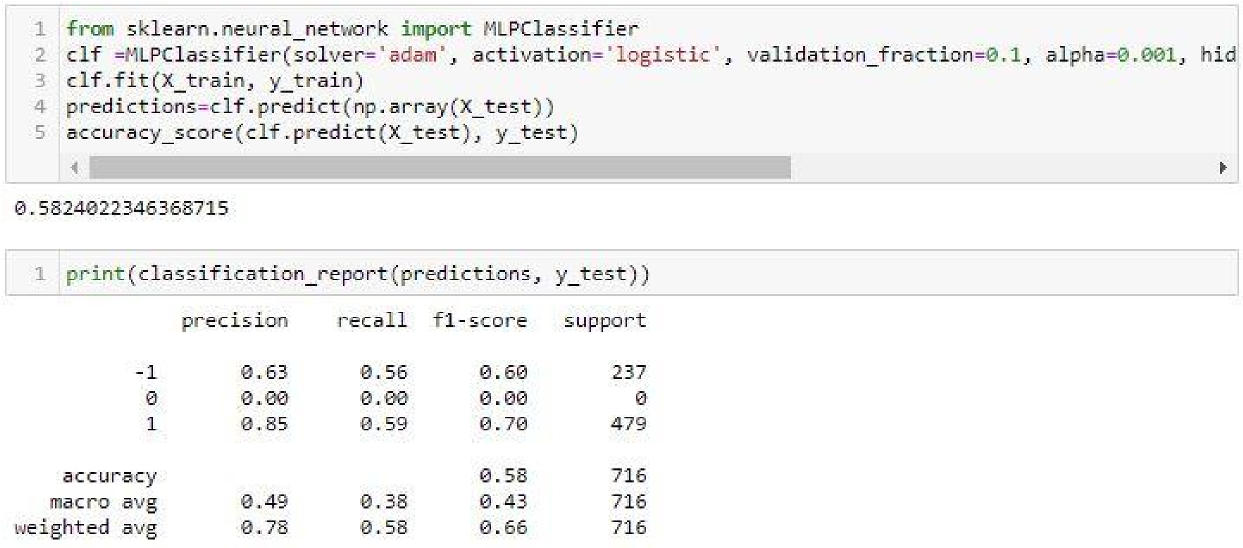
```
from sklearn.svm import SVC
clf=SVC(kernel='rbf')
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.5824022346368715
```

```
print(classification_report(predictions, y_test))
```

```
              precision    recall  f1-score   support

          -1       0.55      0.63      0.59       183
           0       0.01      0.33      0.01         3
           1       0.90      0.57      0.69       530

    accuracy                           0.58       716
   macro avg       0.49      0.51      0.43       716
weighted avg       0.81      0.58      0.66       716
```

1.5.3.5. Neural Network Classifier:



```
from sklearn.neural_network import MLPClassifier
clf =MLPClassifier(solver='adam', activation='logistic', validation_fraction=0.1, alpha=0.001, hid
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

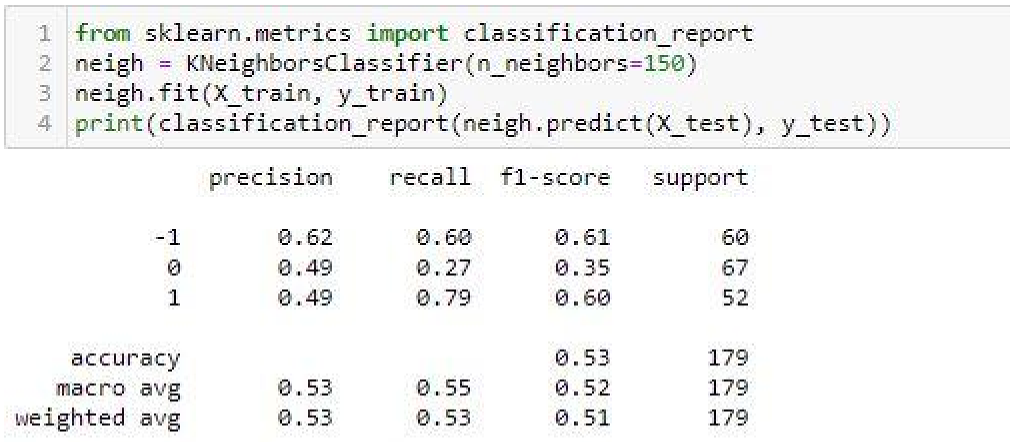
```
0.5824022346368715
```

```
print(classification_report(predictions, y_test))
```

```
              precision    recall  f1-score   support

          -1       0.63      0.56      0.60       237
           0       0.00      0.00      0.00         0
           1       0.85      0.59      0.70       479

    accuracy                           0.58       716
   macro avg       0.49      0.38      0.43       716
weighted avg       0.78      0.58      0.66       716
```

1.5.4 Models Balanced Classes:

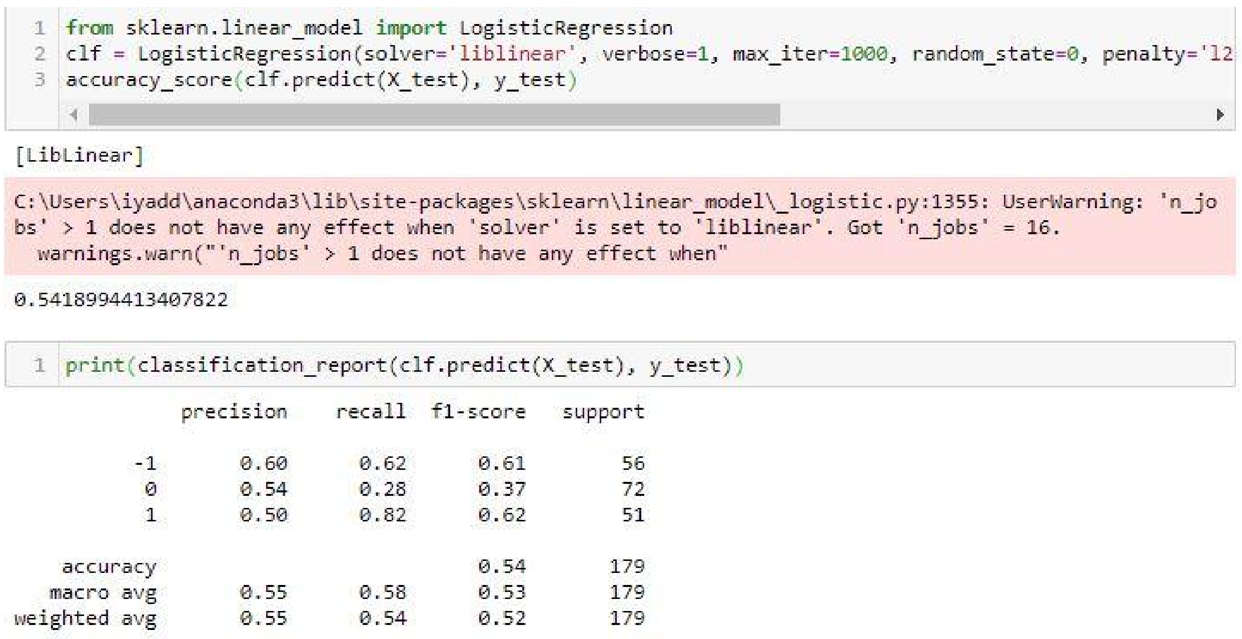
1.5.4.1. K-Nearest Neighbors Classifier:



```
from sklearn.metrics import classification_report
neigh = KNeighborsClassifier(n_neighbors=150)
neigh.fit(X_train, y_train)
print(classification_report(neigh.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.62      0.60      0.61        60
           0       0.49      0.27      0.35        67
           1       0.49      0.79      0.60        52

    accuracy                           0.53       179
   macro avg       0.53      0.55      0.52       179
weighted avg       0.53      0.53      0.51       179
```

1.5.4.2. Logistic Regression Classifier:



```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(solver='liblinear', verbose=1, max_iter=1000, random_state=0, penalty='l2
accuracy_score(clf.predict(X_test), y_test)
```

```
[LibLinear]
```

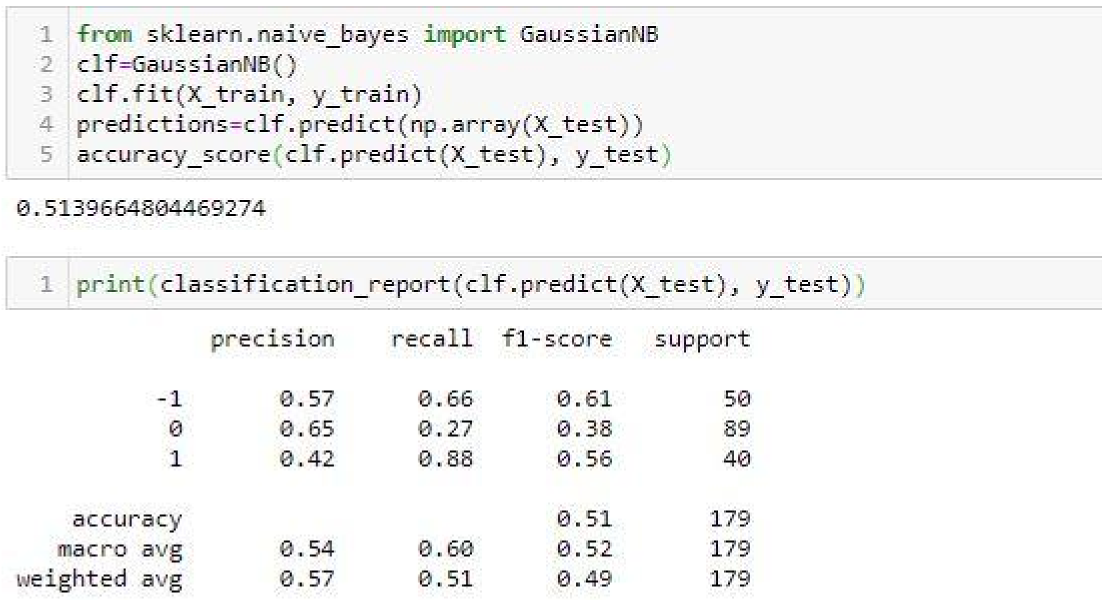
```
C:\Users\iyadd\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1355: UserWarning: 'n_jo
bs' > 1 does not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 16.
  warnings.warn("'n_jobs' > 1 does not have any effect when"
```

```
0.5418994413407822
```

```
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.60      0.62      0.61        56
           0       0.54      0.28      0.37        72
           1       0.50      0.82      0.62        51

    accuracy                           0.54       179
   macro avg       0.55      0.58      0.53       179
weighted avg       0.55      0.54      0.52       179
```

1.5.4.3. Naive Bayes Classifier:



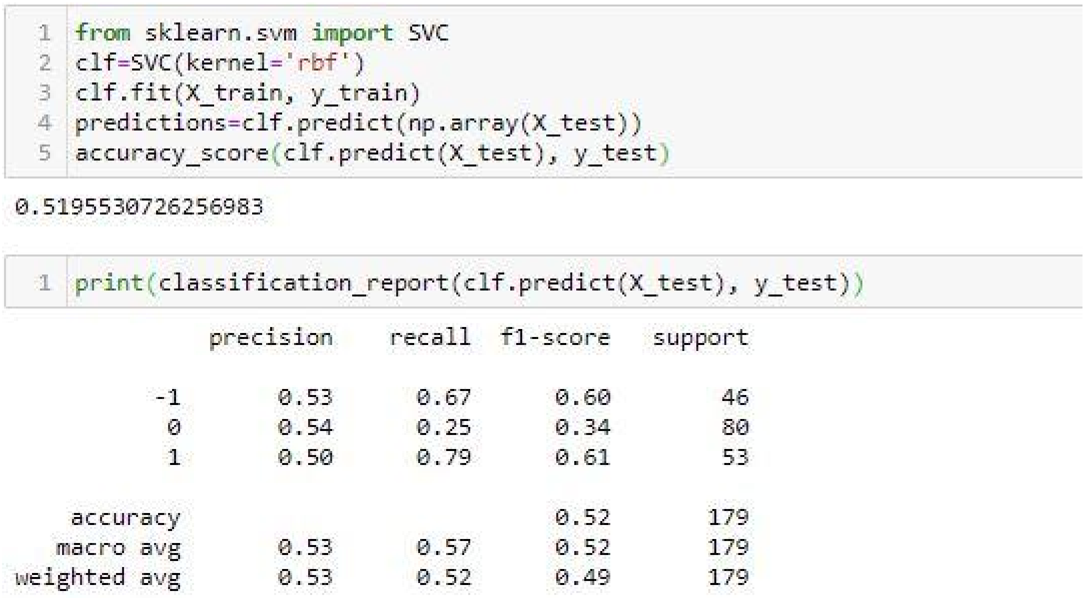
```
from sklearn.naive_bayes import GaussianNB
clf=GaussianNB()
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.5139664804469274
```

```
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.57      0.66      0.61        50
           0       0.65      0.27      0.38        89
           1       0.42      0.88      0.56        40

    accuracy                           0.51       179
   macro avg       0.54      0.60      0.52       179
weighted avg       0.57      0.51      0.49       179
```

1.5.4.4. Support Vector Machine Classifier:



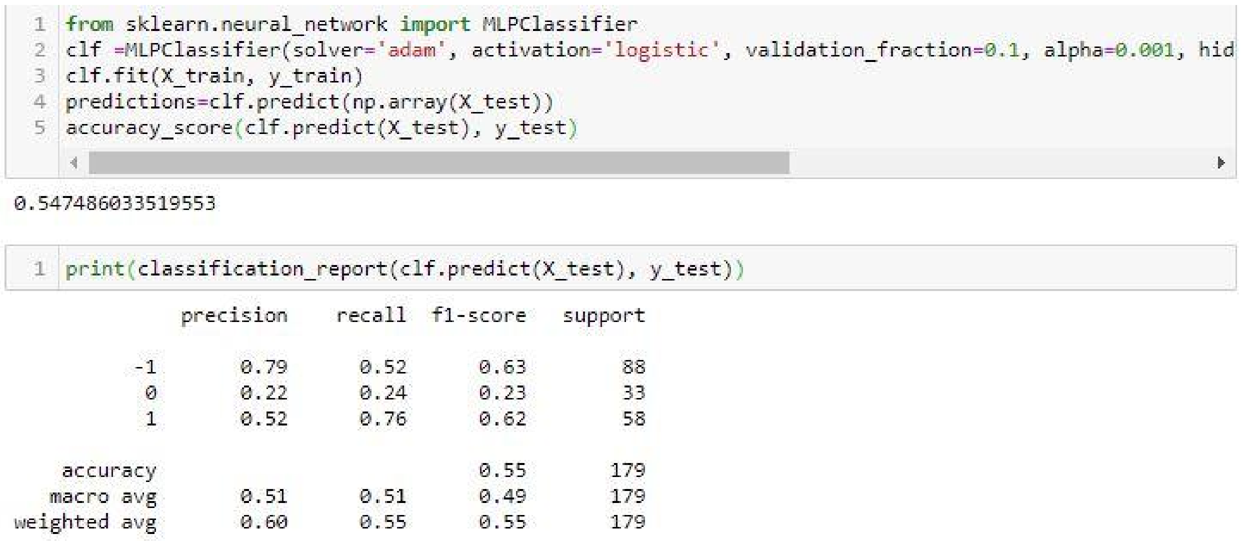
```
from sklearn.svm import SVC
clf=SVC(kernel='rbf')
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.5195530726256983
```

```
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.53      0.67      0.60        46
           0       0.54      0.25      0.34        80
           1       0.50      0.79      0.61        53

    accuracy                           0.52       179
   macro avg       0.53      0.57      0.52       179
weighted avg       0.53      0.52      0.49       179
```

1.5.4.5. Neural Network Classifier:



```
from sklearn.neural_network import MLPClassifier
clf =MLPClassifier(solver='adam', activation='logistic', validation_fraction=0.1, alpha=0.001, hid
clf.fit(X_train, y_train)
predictions=clf.predict(np.array(X_test))
accuracy_score(clf.predict(X_test), y_test)
```

```
0.547486033519553
```

```
print(classification_report(clf.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

          -1       0.79      0.52      0.63        88
           0       0.22      0.24      0.23        33
           1       0.52      0.76      0.62        58

    accuracy                           0.55       179
   macro avg       0.51      0.51      0.49       179
weighted avg       0.60      0.55      0.55       179
```

1.6 Performance Evaluation

As explained above, the best our models were performing when we used a higher number of training samples and while using feature scaling and PCA. The reason for this is that PCA greatly reduces the dimensionality of our data and feature scaling helps PCA capture the variation better as well. We also conclude that if we had more training data our models could have been trained more effectively and thus performed better.

We know that there are various classification metrics that we can derive and use from the confusion matrix including Recall, Precision, Accuracy, and to assess the performance of our model. However, Precision and Recall are useful in cases when the classes aren't evenly distributed, which wasn't the case in our model. And F1 score is preferred in cases when our subject of interest is False Positives and False Negatives. However, As established above, for the performance evaluation of the model we majorly relied on the results of Accuracy as the major purpose of our prediction was to test how closely on overall predictions our model measures the true value and assesses which model best estimated the relationship between given variables in our dataset. However, the accuracy itself sometimes on a standalone basis is not too informative of the effectiveness of our model. Hence, we also used the macro-average method since we wanted to evaluate how our model performs overall across sets of data. Following are the Accuracy and Macro Average results that we obtained from given classifiers:

Evaluation Metric	**Accuracy**				**Macro Average**			
	Without Preprocessing	With Preprocessing	Lesser Training Data	Balanced Classes	Without Preprocessing	With Preprocessing	Lesser Training Data	Balanced Classes
KNN Classifier	0.58	0.61	0.59	0.53	0.45	0.5	0.44	0.52
Logistic Classifier	0.60	0.57	0.57	0.54	0.52	0.52	0.48	0.53
Naive Bayes Classifier	0.51	0.61	0.53	0.51	0.52	0.56	0.42	0.52
SVM	0.49	0.60	0.58	0.52	0.30	0.45	0.43	0.52
MLP Classifier	0.59	0.61	0.58	0.55	0.43	0.45	0.43	0.49

As we can see from the given data, all classifiers have different accuracy scores from other classifiers as the prediction accuracy score is based on hyperparameters and improvement level over other prediction models.

With Preprocessing: The overall accuracies of all the classifiers were improved by identifying and correctly using the optimized slack variables/penalty terms. Thus, the model accuracies derived from KNN, Naive Bayes, SVM, and MLP classifiers are almost close to each other. However, the accuracy derived from Logistic Regression Classifier is relatively lower. Its accuracy is usually affected by outliers and collinearity more than others, thus it has lowest accuracy. Moreover, Neural Networks usually require relatively large dataset for better accuracy, and its 0.61 accuracy implies that the dataset was sufficient enough. Lastly, the 0.61 accuracy score of Naive Bayes Classifier in Preprocessed data is significantly higher than this classifier's scores in Without Preprocessing, Lesser Training data, and Balanced Classes.

Without Preprocessing: While the accuracy scores of other classifiers remain almost the same/close to the accuracy scores in other experiments, SVM's scores are significantly lower.

Lesser Training Data: Only SVM's accuracy score is relatively different while other classifiers' scores under Lesser Training are almost the same than in other methods.

Balanced Classes: KNN, Logistic, and SVM scores are significantly lower than under other methods.

Thus, from the given accuracy scores, we can establish that:

- The accuracy scores of almost all the classifiers under Balanced classes are lower than the accuracy score of the same classifier under any other method. However, all classifiers have almost relatively very close accuracy under the Balanced Classes method.
- Similarly, under Preprocessing method, classifiers accuracy are closer to each other (except LR classifier), but overall accuracy of almost all the classifiers is higher than the accuracy of the same classifier under any method.
- Lastly, in Without Preprocessing, there is more deviation in the accuracy scores

1.7 Conclusion

In conclusion, we started off with raw data of each season of the EPL. From this raw data, we extracted useful features that would tell us about the history of each team. We then used feature scaling and PCA to reduce the dimensionality of the data, which gave us the best accuracies (around 60%). We also found out that due to class imbalance, our models were having a hard time predicting the draw result, and when we balanced the classes, it performed better for the draw class, but worsened on the other classes. We also concluded that a higher amount of training data was improving our results. Thus we believe, having more data would have improved our models significantly. Other than having less data to train on, we were also unable to extract other key features from the raw data that might have an impact on the outcome of a game. Some of them include factors such as the player lineups of the team and player injuries. Including such features would also help our models perform better.