Compare Logistic Regression and Decision Tree in Predicting the football result in the Premier League

Report Outline

Introduction

Football (or soccer) is one of the most popular sports, beloved by billions of people around the world. Every country has its own division league, in which the football clubs will compete with each other to find the national champion. Among all the football leagues, the Premier League – the top level of the English football system – is considered to be the most competitive and exciting. It intrigued viewers by the high quality of the match and also by its unpredictability, where a small team can cause serious trouble for the top-tier clubs. I am also a fan of the league and an avid supporter of the Liverpool Football Club (LFC). This analysis and machine learning model should be useful in predicting the results of Premier League matches.

In this report, I will first put forward the problem and objective in the Problem Formulation section. The method used will be described in the Method part. Then, different results will be presented and discussed in the Results part. The Conclusion section will provide some discussion on the usefulness and validity of different results, the limitations of the model as well as the potential developments.

Problem Formulation

The data points of the problem are the football statistics of every match in a season from the season 2000/2001 to the season 2018/2019. A match statistic represents the numerical value of the following aspects: the match host club (home 1, away 0), the home team goal, away team goal, the difference in goals this season, points of two clubs, etc. The datasets are collected from Kaggle (https://www.kaggle.com/saife245/english-premier-league), a subsidiary of Google LLC, and an online platform for data scientists and machine learning practitioners. The features are factors that can highly impact the outcome of the match, such as the host team, home/away team goal difference, the points of two teams at the time of the match, and the team result in the last 5 games. What I aim to answer, the label, is the result of the future matches whether the host team would win or not.

Method

Dataset

As mentioned above, I collect the data from the Kaggle platform. The match statistics are from the 2000/2001 season to the 2018/2019 season. There are in total 6839 data points with no missing data in any fields. After uploading the data file, I pre-processed it, choosing only the features that I considered highly impactful (justification below) to ensure the accuracy of the model. The get_dummies function in Pandas was also used to convert the categorical values into numerical ones.

Feature justification

- *Host team*: It does matter which team is gonna host the match. Hosting give the club advantage of not having to travel, a familiar playground, and of course, fan encouragement.

- *HTP, ATP*: HTP (Home Team Point) and APT (Away Team Point) indicate the average point that the team earns in that season. This is a good reflection of the corresponding position between the two teams at the moment.
- *HTGD, ATGD*: HTGD (Home Team Goal Difference) and ATDG (Away Team Goal Difference) is the average of the goal the team scored and conceded in that season. In football, the team that scores more goals is the winner; therefore, I believe these two features are strong indicators of the team's winning probability.
- *DiffPts*: Difference of current season points between two teams (home's – away's): The team result can vary significantly among different seasons due to reasons such as changes in line-up. It's important to take into account how the team are performing in this season to predict their future matches
- *DiffFormpPTS (H1 – H5, A1 – A5):* Result of the last 5 matches of the team. This will help us know how the team is performing recently.

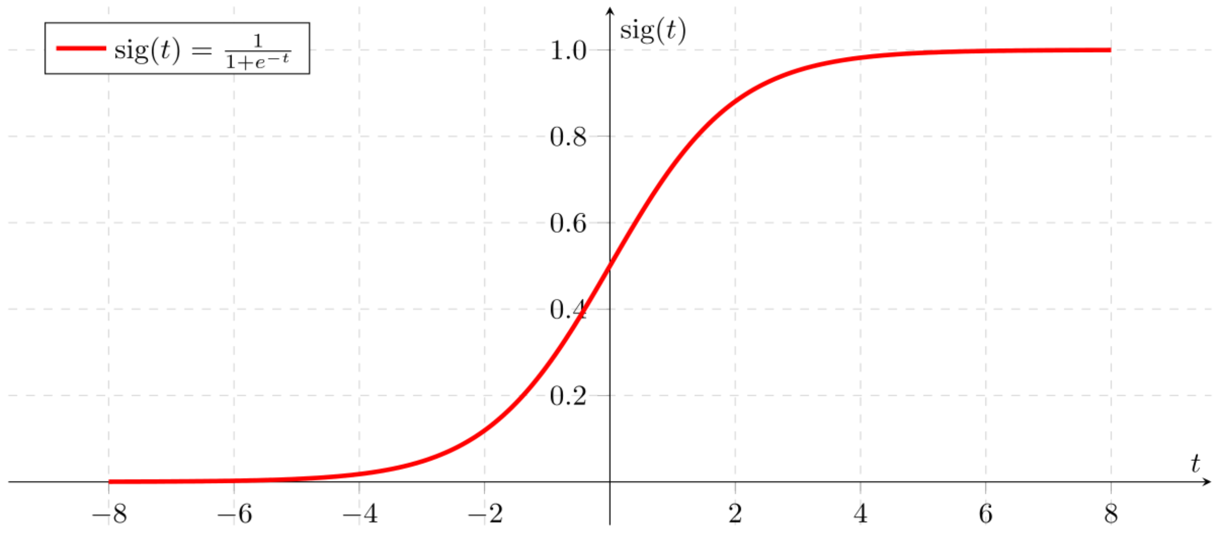
Furthermore, I also visualize the data by using the scatterplot and all of the features show a strong correlation with what we're trying to predict – the 'Result'

Model Validation

The data was split into 3 categories such that 80% is the training set, 10% is the validation test, and 10% is the test set. The Machine Learning model uses quite a lot of features (complicated), so I think it would be best if a large proportion of the data are used as the training set. The remaining are divided into the validation and test set to ensure that the model will provide an unbiased evaluation of the model fit on the dataset.

Logistic Regression Model

My dependent variable (label) is categorical (whether the home team wins or not), Logistic Regression seems to be a rational choice as it is a popular algorithm for solving binary classifications problems. The model uses the Sigmoid function to estimate the value of the label between 0 and 1.



Logistic Loss Function

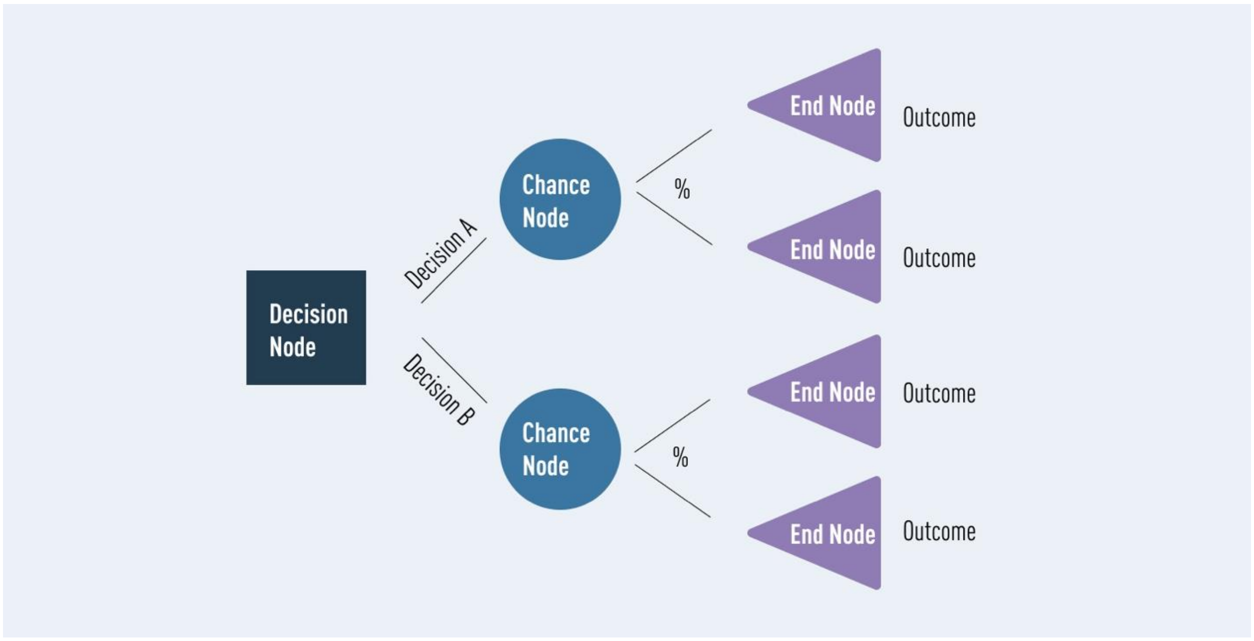
The logistic loss is used to evaluate the prediction of my model:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

I chose to use the logistic loss function as it facilitates the use of an available library for logistic regression and is a strong indicator of how close the prediction is to the corresponding true value (0 and 1 – or whether the host team will win the match or not in my case)

Decision Tree

The second method I decided to use is a Decision Tree Classifier, a commonly used method in binary classification. As my model has numerous features and is complicated, the Decision Tree seems like a reasonable choice as it breaks down complex data into more manageable parts. It is also a good method when we have an important feature in the label that can highly impact the outcome (in my case, it is the host team who are always more probable to win a match). The Decision Tree Classifier normally uses three key nodes to classify the label:



For the Decision Tree Classifier, I also choose logistic loss as it is helpful in predicting the model efficiency in binary classification problem and make it easy for me to compare Decision Tree Classifier with Logistic Regression.

Results

The result (accuracy and loss) for the two machine learning methods are as below:

```
Logistic Regression
```

```
The training accuracy of the model is  0.6465643274853801
The training loss for the model is  12.20735168642486
The validation accuracy of the model is  0.6900584795321637
The validation loss for the model is  10.705114264140049



Decision Tree
The training accuracy of the model is  0.9798976608187134
The training loss for the model is  0.6943116592497226
The validation accuracy of the model is  0.5570175438596491
The validation loss for the model is  15.300242673805181
```

The training and validation accuracy for Logistic Regression is decent, with not so much difference between the accuracies and the losses on the two datasets. Meanwhile, the Decision Tree is superior in training data but the difference between the training and the validation sets are a bit large, suggesting a possibility of over-fitting. Due to the higher accuracy on the validation dataset and a smaller difference between accuracy and loss on the two datasets, I would choose Logistic Regression over Decision Tree for my problem. To examine whether my model evaluation is biased or not, I used the remaining 10% of the datasets as my test set. The test error for my final chosen method (Logistic Regression) is as follow:

```
The test loss for the model is:  11.361547151781407
```

The test error for the Decision Tree Classifier is also computed:

```
The test loss for the model is  14.542806352886698
```

Conclusion

My problem try to predict whether the host team will win or not in a football match in the England Football Division. According to the Results section, I have chosen the Logistic Regression as my final method (justification above) because of the smaller difference between training and validation datasets. This makes sense as due to the simplistic assumption of linear decision boundaries, the Logistic Regression algorithm has been known to be less prone to overfitting.

The result I got from the two models are pretty decent, but I think there is still room for improvement. As the Decision Tree seems to be overfitting, more quality data could be added to increase the efficiency of the model. The model could also utilize more features that are relevant and predictable, such as the location and the time where the match takes place, the match referee, the line-up of two teams. Furthermore, some aspects that happen during the match such as shot on target, fouls, or ball possession, can also be predicted using machine learning and these can then be used as features for our machine learning problem to better predict the outcome of the match.

Reference

[1] Kaggle Dataset: https://www.kaggle.com/saife245/english-premier-league
[2] The Course Book – Chapter 2.3.2 and 3.6: http://mlbook.cs.fi

ML Code

March 22, 2022

[1]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
import pandas.plotting
```

[2]:
```
old_data = pd.read_csv('final_dataset 2.csv')
data = old_data[['FTR','HTGD','ATGD','HTP','ATP','DiffFormPts','DiffPts','HM1',␣
 ↪'HM2','HM3','HM4','HM5','AM1','AM2','AM3','AM4','AM5']]
data.columns =␣
 ↪['Result','HTGD','ATGD','HTP','ATP','DiffFormPts','DiffPts','H1',␣
 ↪'H2','H3','H4','H5','A1','A2','A3','A4','A5']
y = pd.get_dummies(data["Result"])
y = y[['H']]
y = np.ravel(y)
display(data)
print(y)
```

```
     Result      HTGD      ATGD       HTP       ATP  DiffFormPts   DiffPts H1  \
0         H  0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M
1         H  0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M
2        NH  0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M
3        NH  0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M
4         H  0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M
...     ...       ...       ...       ...       ...          ...       ... ..
6835      H -0.289474  0.710526  1.078947  1.842105    -0.263158 -0.763158  L
6836     NH -0.473684  2.052632  0.947368  2.552632    -0.131579 -1.605263  W
6837     NH -0.710526 -0.894737  0.868421  0.789474    -0.052632  0.078947  L
6838      H  0.973684 -0.078947  1.947368  1.236842     0.078947  0.710526  W
6839      H -0.578947 -0.315789  1.026316  1.289474    -0.105263 -0.263158  D

     H2 H3 H4 H5 A1 A2 A3 A4 A5
0     M  M  M  M  M  M  M  M  M
1     M  M  M  M  M  M  M  M  M
2     M  M  M  M  M  M  M  M  M
3     M  M  M  M  M  M  M  M  M
4     M  M  M  M  M  M  M  M  M
...  .. .. .. .. .. .. .. .. ..
6835  L  L  L  W  D  W  W  W  W
```

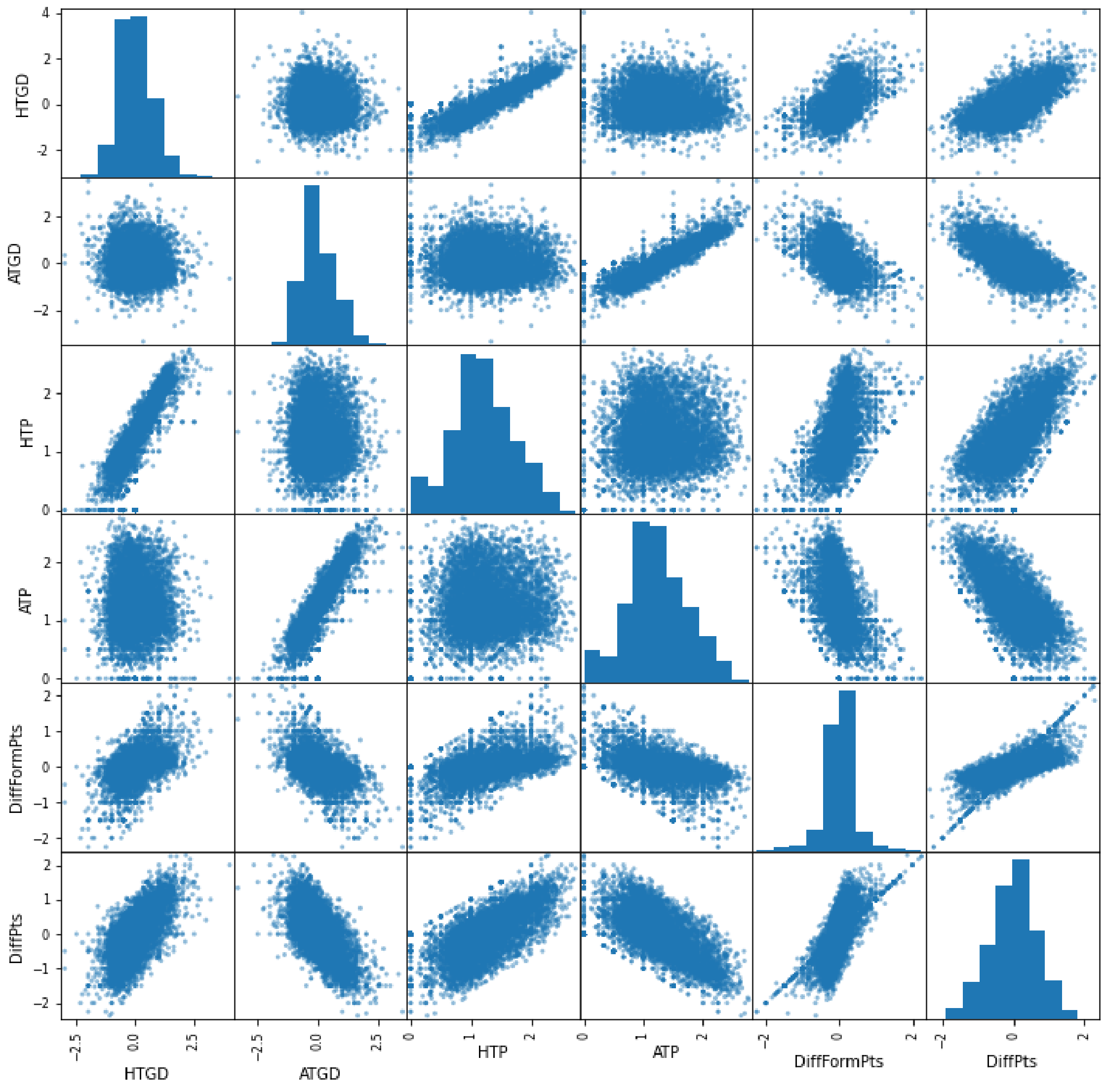
```
6836  D  W  D  L  W  D  W  W  W
6837  L  L  L  D  L  D  D  D  L
6838  L  W  D  L  W  L  L  D  L
6839  W  L  L  D  D  W  W  D  D

[6840 rows x 17 columns]


[1 1 0 … 0 1 1]
```

[3]:
```
# To check whether the features I chose have enough correlation with the label␣
↪(FTR - Full Time Result)
pd.plotting.
↪scatter_matrix(data[['HTGD','ATGD','HTP','ATP','DiffFormPts','DiffPts']],␣
↪figsize=(12, 12))
```

[3]:
```
array([[<AxesSubplot:xlabel='HTGD', ylabel='HTGD'>,
        <AxesSubplot:xlabel='ATGD', ylabel='HTGD'>,
        <AxesSubplot:xlabel='HTP', ylabel='HTGD'>,
        <AxesSubplot:xlabel='ATP', ylabel='HTGD'>,
        <AxesSubplot:xlabel='DiffFormPts', ylabel='HTGD'>,
        <AxesSubplot:xlabel='DiffPts', ylabel='HTGD'>],
       [<AxesSubplot:xlabel='HTGD', ylabel='ATGD'>,
        <AxesSubplot:xlabel='ATGD', ylabel='ATGD'>,
        <AxesSubplot:xlabel='HTP', ylabel='ATGD'>,
        <AxesSubplot:xlabel='ATP', ylabel='ATGD'>,
        <AxesSubplot:xlabel='DiffFormPts', ylabel='ATGD'>,
        <AxesSubplot:xlabel='DiffPts', ylabel='ATGD'>],
       [<AxesSubplot:xlabel='HTGD', ylabel='HTP'>,
        <AxesSubplot:xlabel='ATGD', ylabel='HTP'>,
        <AxesSubplot:xlabel='HTP', ylabel='HTP'>,
        <AxesSubplot:xlabel='ATP', ylabel='HTP'>,
        <AxesSubplot:xlabel='DiffFormPts', ylabel='HTP'>,
        <AxesSubplot:xlabel='DiffPts', ylabel='HTP'>],
       [<AxesSubplot:xlabel='HTGD', ylabel='ATP'>,
        <AxesSubplot:xlabel='ATGD', ylabel='ATP'>,
        <AxesSubplot:xlabel='HTP', ylabel='ATP'>,
        <AxesSubplot:xlabel='ATP', ylabel='ATP'>,
        <AxesSubplot:xlabel='DiffFormPts', ylabel='ATP'>,
        <AxesSubplot:xlabel='DiffPts', ylabel='ATP'>],
       [<AxesSubplot:xlabel='HTGD', ylabel='DiffFormPts'>,
        <AxesSubplot:xlabel='ATGD', ylabel='DiffFormPts'>,
        <AxesSubplot:xlabel='HTP', ylabel='DiffFormPts'>,
        <AxesSubplot:xlabel='ATP', ylabel='DiffFormPts'>,
        <AxesSubplot:xlabel='DiffFormPts', ylabel='DiffFormPts'>,
        <AxesSubplot:xlabel='DiffPts', ylabel='DiffFormPts'>],
       [<AxesSubplot:xlabel='HTGD', ylabel='DiffPts'>,
        <AxesSubplot:xlabel='ATGD', ylabel='DiffPts'>,
        <AxesSubplot:xlabel='HTP', ylabel='DiffPts'>,
        <AxesSubplot:xlabel='ATP', ylabel='DiffPts'>,
        <AxesSubplot:xlabel='DiffFormPts', ylabel='DiffPts'>,
        <AxesSubplot:xlabel='DiffPts', ylabel='DiffPts'>]], dtype=object)
```



[5]:
```
# Create the feature for the dataset
X = data.drop(['Result'],1)
print (X)

# Create the label (Result) for the dataset
```

```
          HTGD      ATGD       HTP       ATP  DiffFormPts   DiffPts H1  H2  H3  \
0     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M   M   M
1     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M   M   M
```

```
2      0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M  M  M
3      0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M  M  M
4      0.000000  0.000000  0.000000  0.000000     0.000000  0.000000  M  M  M
…           …         …         …         …            …        … .. .. ..
6835 -0.289474  0.710526  1.078947  1.842105    -0.263158 -0.763158  L  L  L
6836 -0.473684  2.052632  0.947368  2.552632    -0.131579 -1.605263  W  D  W
6837 -0.710526 -0.894737  0.868421  0.789474    -0.052632  0.078947  L  L  L
6838  0.973684 -0.078947  1.947368  1.236842     0.078947  0.710526  W  L  W
6839 -0.578947 -0.315789  1.026316  1.289474    -0.105263 -0.263158  D  W  L

     H4 H5 A1 A2 A3 A4 A5
0     M  M  M  M  M  M  M
1     M  M  M  M  M  M  M
2     M  M  M  M  M  M  M
3     M  M  M  M  M  M  M
4     M  M  M  M  M  M  M
…    .. .. .. .. .. .. ..
6835  L  W  D  W  W  W  W
6836  D  L  W  D  W  W  W
6837  L  D  L  D  D  D  L
6838  D  L  W  L  L  D  L
6839  L  D  D  W  W  D  D

[6840 rows x 16 columns]
```

[6]:
```
X_new = 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)           #␣
 ↪Convert categorical into dummies values
    X_new = X_new.join(col_data)
print(X_new)
X = X_new
print(X)
```

```
          HTGD      ATGD       HTP       ATP  DiffFormPts   DiffPts  H1_D  \
0     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
1     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
2     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
3     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
4     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
…           …         …         …         …            …        …   …
6835 -0.289474  0.710526  1.078947  1.842105    -0.263158 -0.763158     0
6836 -0.473684  2.052632  0.947368  2.552632    -0.131579 -1.605263     0
6837 -0.710526 -0.894737  0.868421  0.789474    -0.052632  0.078947     0
6838  0.973684 -0.078947  1.947368  1.236842     0.078947  0.710526     0
6839 -0.578947 -0.315789  1.026316  1.289474    -0.105263 -0.263158     1
```

```
      H1_L  H1_M  H1_W  …  A3_M  A3_W  A4_D  A4_L  A4_M  A4_W  A5_D  A5_L  \
0        0     1     0  …     1     0     0     0     1     0     0     0
1        0     1     0  …     1     0     0     0     1     0     0     0
2        0     1     0  …     1     0     0     0     1     0     0     0
3        0     1     0  …     1     0     0     0     1     0     0     0
4        0     1     0  …     1     0     0     0     1     0     0     0
…    …     …     …  …     …     …     …     …     …     …     …     …
6835     1     0     0  …     0     1     0     0     0     1     0     0
6836     0     0     1  …     0     1     0     0     0     1     0     0
6837     1     0     0  …     0     0     1     0     0     0     0     1
6838     0     0     1  …     0     0     1     0     0     0     0     1
6839     0     0     0  …     0     1     1     0     0     0     1     0

      A5_M  A5_W
0        1     0
1        1     0
2        1     0
3        1     0
4        1     0
…    …     …
6835     0     1
6836     0     1
6837     0     0
6838     0     0
6839     0     0

[6840 rows x 46 columns]
          HTGD      ATGD       HTP       ATP  DiffFormPts   DiffPts  H1_D  \
0     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
1     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
2     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
3     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
4     0.000000  0.000000  0.000000  0.000000     0.000000  0.000000     0
…          …         …         …         …            …         …   …
6835 -0.289474  0.710526  1.078947  1.842105    -0.263158 -0.763158     0
6836 -0.473684  2.052632  0.947368  2.552632    -0.131579 -1.605263     0
6837 -0.710526 -0.894737  0.868421  0.789474    -0.052632  0.078947     0
6838  0.973684 -0.078947  1.947368  1.236842     0.078947  0.710526     0
6839 -0.578947 -0.315789  1.026316  1.289474    -0.105263 -0.263158     1

      H1_L  H1_M  H1_W  …  A3_M  A3_W  A4_D  A4_L  A4_M  A4_W  A5_D  A5_L  \
0        0     1     0  …     1     0     0     0     1     0     0     0
1        0     1     0  …     1     0     0     0     1     0     0     0
2        0     1     0  …     1     0     0     0     1     0     0     0
3        0     1     0  …     1     0     0     0     1     0     0     0
4        0     1     0  …     1     0     0     0     1     0     0     0
…    …     …     …  …     …     …     …     …     …     …     …     …
6835     1     0     0  …     0     1     0     0     0     1     0     0
```

```
6836      0      0      1  …      0      1      0      0      0      1      0      0
6837      1      0      0  …      0      0      1      0      0      0      0      1
6838      0      0      1  …      0      0      1      0      0      0      0      1
6839      0      0      0  …      0      1      1      0      0      0      1      0

      A5_M  A5_W
0        1     0
1        1     0
2        1     0
3        1     0
4        1     0
…     …    …
6835     0     1
6836     0     1
6837     0     0
6838     0     0
6839     0     0

[6840 rows x 46 columns]
```

[11]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss

X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size = 0.5,␣
 ↪random_state = 42)

clf = LogisticRegression(C=10000)

clf.fit(X_train, y_train)
y_pred_train = clf.predict(X_train)
acc_train1 = accuracy_score(y_train, y_pred_train)
loss1 = log_loss(y_train, y_pred_train)
print("Logistic Regression")
print("The training accuracy of the model is ", acc_train1)
print("The training loss for the model is ", loss1)


clf.fit(X_val, y_val)
y_pred_val = clf.predict(X_val)
loss2 = log_loss(y_val, y_pred_val)
acc_train2 = accuracy_score(y_val, y_pred_val)
print("The validation accuracy of the model is ", acc_train2)
print("The validation loss for the model is ", loss2)
```

```
clf.fit(X_test, y_test)
y_pred_test = clf.predict(X_test)
loss3 = log_loss(y_test, y_pred_test)
acc_train3 = accuracy_score(y_test, y_pred_test)
print("The test accuracy of the model is ", acc_train3)
print("The test loss for the model is: ", loss3)
```

```
Logistic Regression
The training accuracy of the model is  0.6465643274853801
The training loss for the model is  12.20735168642486
The validation accuracy of the model is  0.6900584795321637
The validation loss for the model is  10.705114264140049
The test accuracy of the model is  0.6710526315789473
The test loss for the model is:  11.361547151781407
```

[12]:
```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
clf2 = DecisionTreeClassifier(random_state=0)
clf2.fit(X_train, y_train)
y_pred_train = clf2.predict(X_train)
acc_train1 = accuracy_score(y_train, y_pred_train)
loss1 = log_loss(y_train, y_pred_train)
print("Decision Tree")
print("The training accuracy of the model is ", acc_train1)
print("The training loss for the model is ", loss1)

y_pred_val = clf2.predict(X_val)
acc_train2 = accuracy_score(y_val, y_pred_val)
loss2 = log_loss(y_val, y_pred_val)
print("The validation accuracy of the model is ", acc_train2)
print("The validation loss for the model is ", loss2)

y_pred_test = clf2.predict(X_test)
acc_train3 = accuracy_score(y_test, y_pred_test)
loss3 = log_loss(y_test, y_pred_test)
print("The test accuracy of the model is ", acc_train3)
print("The test loss for the model is ", loss3)
```

```
Decision Tree
The training accuracy of the model is  0.9798976608187134
The training loss for the model is  0.6943116592497226
The validation accuracy of the model is  0.5570175438596491
The validation loss for the model is  15.300242673805181
The test accuracy of the model is  0.5789473684210527
The test loss for the model is  14.542806352886698
```