CPT_S 534: Neural Network Design & Application **– Project Final Report**
05/04/2022

Team:
011717650 - Sahil Shrivastava
011738571 - Andrei Kondyrev
011771817 - Kulpreet Singh

Introduction:

Initially our project was on age detection of a person using deep learning analysis. However, the three of us have an undeniable passion towards football (or soccer). We thus decided to pivot our focus towards detecting the winners of a football league based on the data given to a neural network which would efficiently classify and predict the teams that have either the chance to lose or win. Massive amounts of statistical data are produced when a football match occurs. This data can be utilized and be fed into a deep learning algorithm to yield perdition results. Our plan is to utilize multiple deep learning algorithms to see which neural network performs the best. In addition, we have focused on involving the English Premier League.

The main application or useability is not only for prediction in wins or losses but also for those who are involved in the Betting realm. Betting is a growing industry around the world, with FIFA estimating that the total global turnover from football tournament betting is worth billions of dollars as stated by the official FIFA World Cup Organization, and the realm stays within the rules and regulations of FIFA.

Literature Survey:

In today's world, both football experts and machines are used to predict the outcomes of football matches. Football as a sport generates a large amount of statistical data about its players, matches between teams, and the environment in which the game is played. This statistical data can be used to predict various information related to a specific football match, such as the game's outcome, a player's injury, performance in a specific match, spotting new talents in the game, and so on, using various machine learning techniques. Moreover, many research articles have been localized to smaller leagues. Initially we planned on utilizing the Recurring Neural Network (RNN).

RNNs theoretically outperform traditional machine learning algorithms. As a result, in addition to the more common applications the outcome of sporting events has yielded promising results. Using a broader set of attributes and including Individual player statistics help predict a player's form from season to season. Larger datasets will also aid in the training of the neural network. Because football is such a popular and widely practiced sport, predicting match results is a fascinating challenge. Predicting the outcomes, on the other hand, is a difficult problem due to the large number of factors that must be considered, many of which cannot be quantified or modeled.

However, due to the large size of the dataset, the RNN displayed enormous computation complexities with extremely long processing times, resulting in an overall network crash. As a result, we shifted our focus to a feedforward neural network, which has proven to be the most effective in any classification-based research. For our project, we used two types of FeedForward Networks, one with a single hidden layer and the other with two hidden layers. Both networks produced excellent accuracies while requiring minimal processing time. The outcome of a match is determined by a number of factors, necessitating a large number of tests to identify the best subset with the greatest impact on the match's final outcome. As a result, the use of these factors are excellent qualitative traits to feed into the neural network, yielding good classification results.

Finally, after the data was trained and the testing prediction rates were collected, the trained data was parsed into three betting strategies: flat, martingale, and 2-6. We looked at the profit returns from different techniques and plotted them, and we found some quite intriguing results.

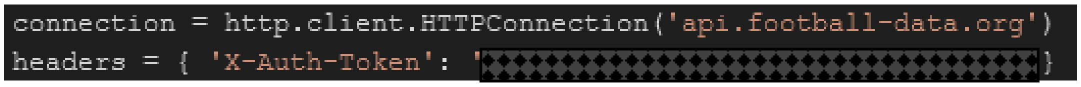
Dataset:

Initial Dataset Utilized:

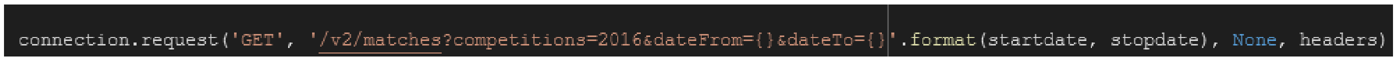
The dataset utilized was an API.

- *api.football-data.org*

To parse the data into the algorithm is an Software called **Postman Software,** wherein we then pass it in by using the “GET” function.
A token has to be purchased for this dataset thus It is slightly confidential, since we needed to purchase the data.

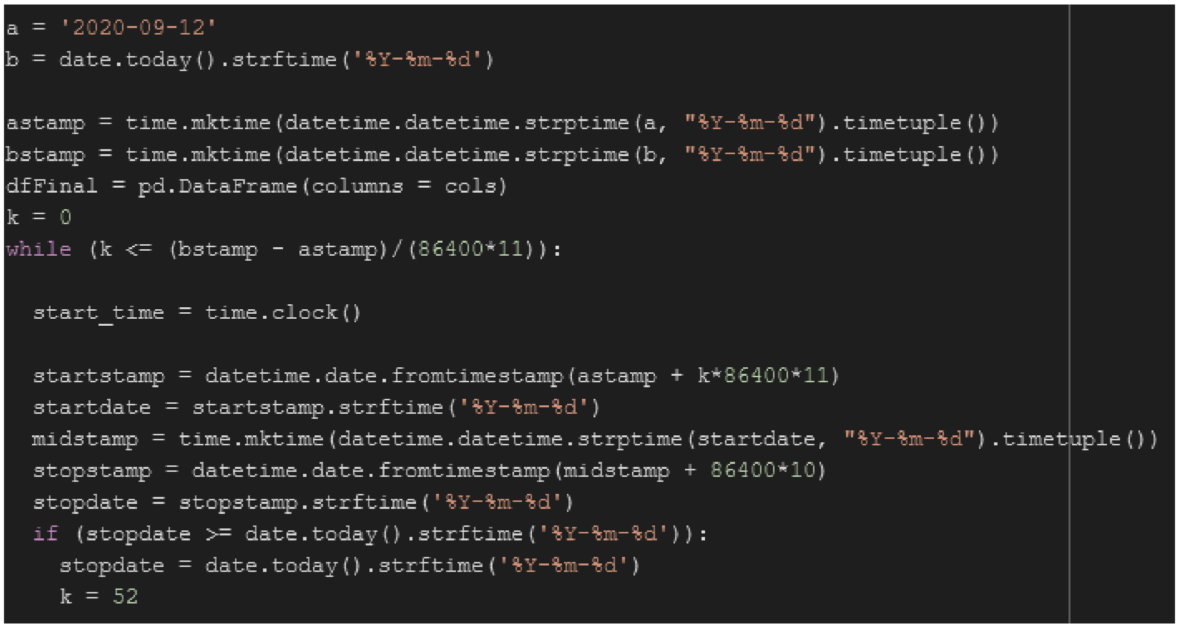


```
connection = http.client.HTTPConnection('api.football-data.org')
headers = { 'X-Auth-Token': '[redacted]'}
```



```
connection.request('GET', '/v2/matches?competitions=2016&dateFrom={}&dateTo={}'.format(startdate, stopdate), None, headers)
```

Range of Data:
Championship League
Start Date: **2020 – 09- 12**
End Date: **Current Date of Today**
Postman Software only allows to parse the range of football matches for a span of 10 days.
Thus, a loop was utilized to parse it by 10 days each



```
a = '2020-09-12'
b = date.today().strftime('%Y-%m-%d')

astamp = time.mktime(datetime.datetime.strptime(a, "%Y-%m-%d").timetuple())
bstamp = time.mktime(datetime.datetime.strptime(b, "%Y-%m-%d").timetuple())
dfFinal = pd.DataFrame(columns = cols)
k = 0
while (k <= (bstamp - astamp)/(86400*11)):

  start_time = time.clock()

  startstamp = datetime.date.fromtimestamp(astamp + k*86400*11)
  startdate = startstamp.strftime('%Y-%m-%d')
  midstamp = time.mktime(datetime.datetime.strptime(startdate, "%Y-%m-%d").timetuple())
  stopstamp = datetime.date.fromtimestamp(midstamp + 86400*10)
  stopdate = stopstamp.strftime('%Y-%m-%d')
  if (stopdate >= date.today().strftime('%Y-%m-%d')):
    stopdate = date.today().strftime('%Y-%m-%d')
    k = 52
```

Updated Dataset:

Originally, the previous dataset did not suit our needs for our project since the features utilized in that dataset were not efficient in calculating the prediction rate of winners/losses of a match.

Thus came the need to switch to another dataset i.e. from the Oddsportal website.

We parsed it because it hosted a larger variety of statistically useful data to predict a winner of a match.

The dataset is:

- Parsed data from *oddsportal.com* for *English Premier League*.

Baseline and Main Approach:

In the world of football, especially when it comes to predicting the winner or loser of a match, a high accuracy rate isn't required to achieve excellent prediction results.

Analysis:

- Consider the following two teams: Team A and Team B:

- The odds of a win-draw in an ideal world scenario are: 0-HOME TEAM win, 1-draw, 2-AWAY TEAM win.

- Now, if Team A has a 3.89 chance of winning, Team B has a 2.10 chance of winning, and the chance of it being a draw is 3.37, then Team A has a 100/3.89 = 25.7 percent chance of winning, Team B has a 100/2.10 = 47.62 percent chance of winning, and the probability of it being a draw is 100/3.37 = 29.67 percent. 25.7 percent + 47.62 percent + 29.67 percent = 102.99 percent 103 percent.

- As a result, there is a 3% margin. Because of this 3% margin, if a statistical analysis is performed and the odds of winning are less than 50%+bookmaker's margin, it's impossible to win money. If the odds are around 55% or greater, it means that the betting player will have profit. As previously stated, the basic idea is to evaluate the match's winners and losers using various evaluation metrics.

- The abscissa is made up of metrics/features, while the ordinate label is made up of the win/loss feature (classification label). The training process will then quantitatively take into account and analyze the matches that are being held, taking into account many statistical attributes that will be noted down in the Evaluation Metrics section below.

Following the evaluation of these attributes, testing will be conducted on a completely new set of data in order to predict whether the team will win or lose a match based on the information provided.

Here is a sample pseudocode:

```
1. Import the necessary libraries
2. Either using TensorFlow/pytorch import the learning library
3. Initially - Imported the data by parsing it from the Postman Software (via the purchased token).
   Updated: Parsed data from the oddsportal.com website.
4. Define the attributes and perform data cleaning.
5. Create the Neural Network based on the given information
6. Feed the system with data.
7. Train the model using the function created in step 5.
8. Calculate the bettor's profit using 3 most popular betting strategies.
```

Finally, we will obtain the classification results regarding the status of the team victory or loss.
Additional Steps:

- Append the teams to the array by parsing them.
- Add these appending arrays to the array to make it bigger.
- Add 11 players to the team because the team size is 11.
- A small intermediate dataframe that will be used to convert the appended array into a single row of the dataframe.

- Input the data into a Neural Network and train it.
- After the model has been trained, the classification accuracy results will be displayed and calculated, and we will be able to determine the match winner/loser by looking at these numbers.

For the updated dataset we skip some of these steps. We just need to clean the data be dropping unneded columns we got after parsing the website.

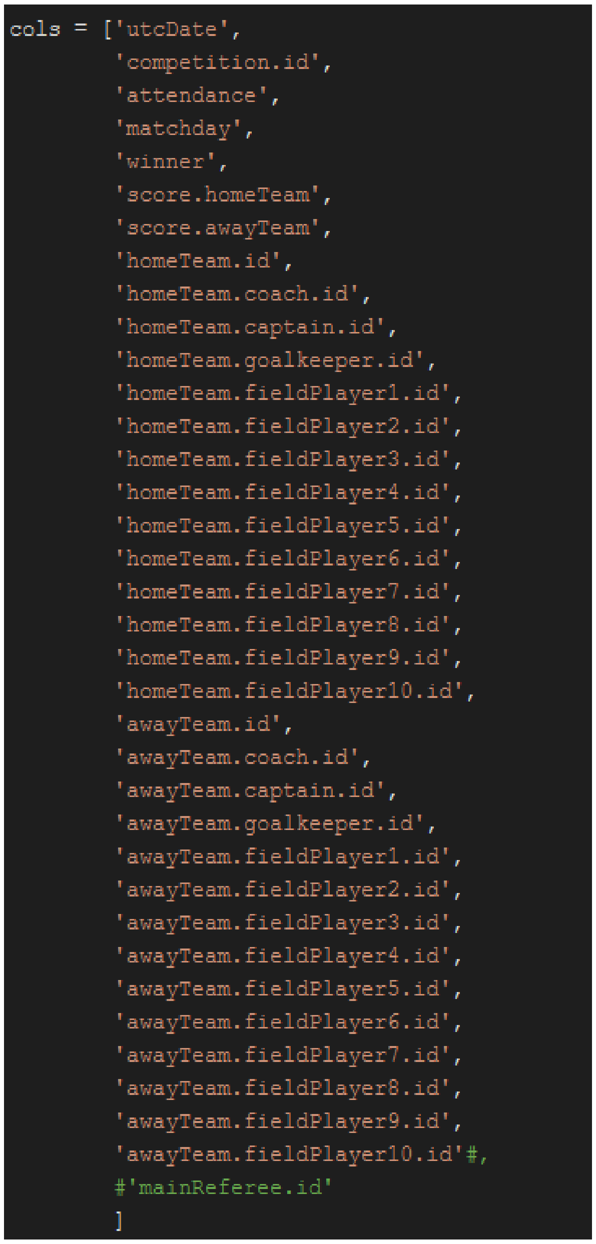
Evaluation Metric:

Initial Dataset	Updataed Dataset
Match Day – If the match takes place closer to the championship, the team may perform better.	*Season* – Presents the year when the match took place.
Winner	*Winner*
Home team ID	*home_team_cat*
Away team ID	*away_team_cat*
Captain ID (for both Home and Away Team) – The captain is crucial because they are the ones who assist and motivate the junior players to improve their game.	Not Applicable for this dataset
Coach ID (for both the Home and Away Teams) – A crucial factor because a good coach can lead a team to victory even if the players on the team are relatively inexperienced.	
Goalkeeper – An important part of training because the goalkeeper determines whether the team wins or loses. A good goalkeeper equals a winning team.	
Full Lineup (Both Home and Away Time)	
Game Score – The winner may be determined by the team's morale rather than the overall score.	

Updated Dataset Features Continued:

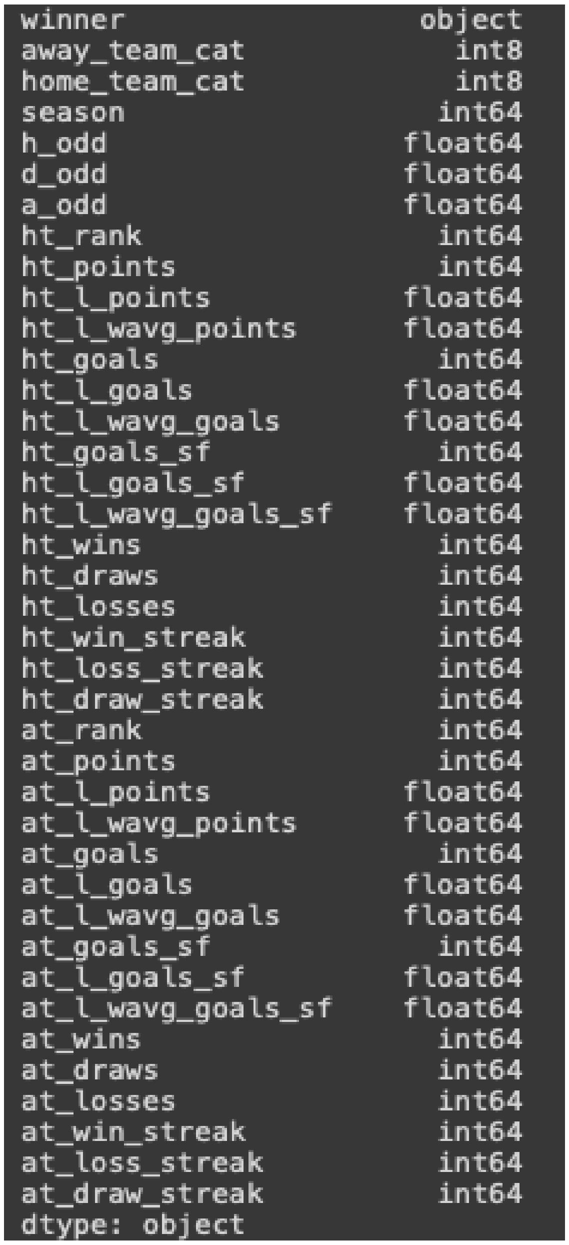
- *h_odd* – odd for the home team to win
- *d_odd* – odds for the teams to get a draw
- *a_odd* – odds for the away team to win
- *ht_rank* – Rank of home team by the time of the specific match

- *ht_points* - Number of points the home team has before the start of the current match
- *ht_l_points*
- *ht_l_wavg_points*
- *ht_goals* - Number of goals scored (in whole in the current years league) by the home team before the start of the match
- *ht_l_goals*
- *ht_l_wavg_goals*
- *ht_goals_sf*
- *ht_l_goals_sf*
- *ht_l_wavg_goals_sf*
- *ht_wins* – Number of wins in the current year league by the home team before the start of the match
- *ht_draws* - Number of draws in the current year league by the home team before the start of the match
- *ht_losses*- Number of losses in the current year league by the home team before the start of the match
- *ht_win_streak* - Consecutive number of the win streak of the home team before the start of that specific match.
- *ht_loss_streak* - Consecutive number of the losses streak of the home team before the start of that specific match.
- *ht_draw_streak* - Consecutive number of the draws streak of the home team before the start of that specific match.
- *at_rank* - Rank of away team by the time of the specific match
- *at_points*- Number of points the away team has before the start of the current match
- *at_l_points*
- *at_l_wavg_points*
- *at_goals* - Number of goals scored (in whole in the current years league) by the away team before the start of the match
- *at_l_goals*
- *at_l_wavg_goals*
- *at_goals_sf*
- *at_l_goals_sf*
- *at_l_wavg_goals_sf*
- *at_wins* - Number of wins in the current year league by the away team before the start of the match
- *at_draws* - Number of draws in the current year league by the away team before the start of the match
- *at_losses* - Number of losses in the current year league by the away team before the start of the match
- *at_win_streak* - Consecutive number of the win streak of the away team before the start of that specific match.
- *at_loss_streak* - Consecutive number of the losses streak of the away team before the start of that specific match.
- *at_draw_streak* - Consecutive number of the draws streak of the away team before the start of that specific match.



```
cols = ['utcDate',
        'competition.id',
        'attendance',
        'matchday',
        'winner',
        'score.homeTeam',
        'score.awayTeam',
        'homeTeam.id',
        'homeTeam.coach.id',
        'homeTeam.captain.id',
        'homeTeam.goalkeeper.id',
        'homeTeam.fieldPlayer1.id',
        'homeTeam.fieldPlayer2.id',
        'homeTeam.fieldPlayer3.id',
        'homeTeam.fieldPlayer4.id',
        'homeTeam.fieldPlayer5.id',
        'homeTeam.fieldPlayer6.id',
        'homeTeam.fieldPlayer7.id',
        'homeTeam.fieldPlayer8.id',
        'homeTeam.fieldPlayer9.id',
        'homeTeam.fieldPlayer10.id',
        'awayTeam.id',
        'awayTeam.coach.id',
        'awayTeam.captain.id',
        'awayTeam.goalkeeper.id',
        'awayTeam.fieldPlayer1.id',
        'awayTeam.fieldPlayer2.id',
        'awayTeam.fieldPlayer3.id',
        'awayTeam.fieldPlayer4.id',
        'awayTeam.fieldPlayer5.id',
        'awayTeam.fieldPlayer6.id',
        'awayTeam.fieldPlayer7.id',
        'awayTeam.fieldPlayer8.id',
        'awayTeam.fieldPlayer9.id',
        'awayTeam.fieldPlayer10.id'#,
        #'mainReferee.id'
        ]
```

Snippet of our code on the top, portrays the evaluation metric being fed into the column of the original dataset matrix.



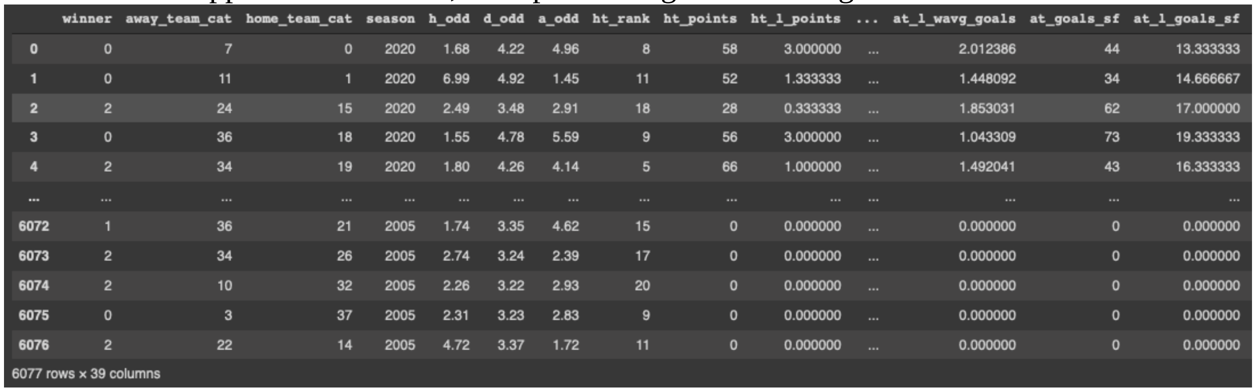
```
winner                  object
away_team_cat             int8
home_team_cat             int8
season                   int64
h_odd                  float64
d_odd                  float64
a_odd                  float64
ht_rank                  int64
ht_points                int64
ht_l_points            float64
ht_l_wavg_points       float64
ht_goals                 int64
ht_l_goals             float64
ht_l_wavg_goals        float64
ht_goals_sf              int64
ht_l_goals_sf          float64
ht_l_wavg_goals_sf     float64
ht_wins                  int64
ht_draws                 int64
ht_losses                int64
ht_win_streak            int64
ht_loss_streak           int64
ht_draw_streak           int64
at_rank                  int64
at_points                int64
at_l_points            float64
at_l_wavg_points       float64
at_goals                 int64
at_l_goals             float64
at_l_wavg_goals        float64
at_goals_sf              int64
at_l_goals_sf          float64
at_l_wavg_goals_sf     float64
at_wins                  int64
at_draws                 int64
at_losses                int64
at_win_streak            int64
at_loss_streak           int64
at_draw_streak           int64
dtype: object
```

Snippet of our code on the top, portrays the evaluation metric being fed into the column of the updated dataset matrix.

Result & Analysis:

1. *Data Cleaning:*

Below is the snipped of the dataset, after performing data cleaning.



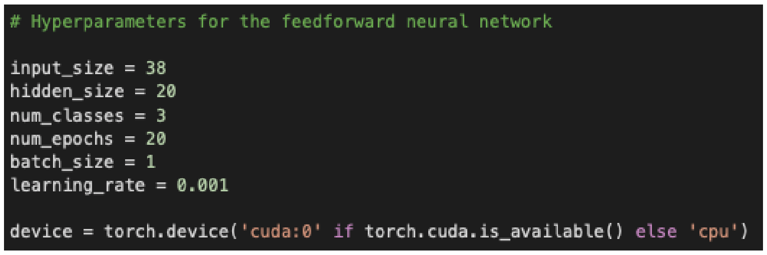
	winner	away_team_cat	home_team_cat	season	h_odd	d_odd	a_odd	ht_rank	ht_points	ht_l_points	...	at_l_wavg_goals	at_goals_sf	at_l_goals_sf
0	0	7	0	2020	1.68	4.22	4.96	8	58	3.000000	...	2.012386	44	13.333333
1	0	11	1	2020	6.99	4.92	1.45	11	52	1.333333	...	1.448092	34	14.666667
2	2	24	15	2020	2.49	3.48	2.91	18	28	0.333333	...	1.853031	62	17.000000
3	0	36	18	2020	1.55	4.78	5.59	9	56	3.000000	...	1.043309	73	19.333333
4	2	34	19	2020	1.80	4.26	4.14	5	66	1.000000	...	1.492041	43	16.333333
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6072	1	36	21	2005	1.74	3.35	4.62	15	0	0.000000	...	0.000000	0	0.000000
6073	2	34	26	2005	2.74	3.24	2.39	17	0	0.000000	...	0.000000	0	0.000000
6074	2	10	32	2005	2.26	3.22	2.93	20	0	0.000000	...	0.000000	0	0.000000
6075	0	3	37	2005	2.31	3.23	2.83	9	0	0.000000	...	0.000000	0	0.000000
6076	2	22	14	2005	4.72	3.37	1.72	11	0	0.000000	...	0.000000	0	0.000000

6077 rows × 39 columns

- Removed the column which had an N/A's or Null Values.
- Removed the columns which implicitly showed which team won that specific match.
- Encoded the names of the teams to categorize the teams in numbers rather than "string" names.
- Removed unnecessary columns that will not aid in prediction ex. match_name, date.
- In the winner's columns, we changed the labels of home_team, draw, away_team, to 0,1,2 respectively.
- Moved the winner column to the first index in the dataframe.

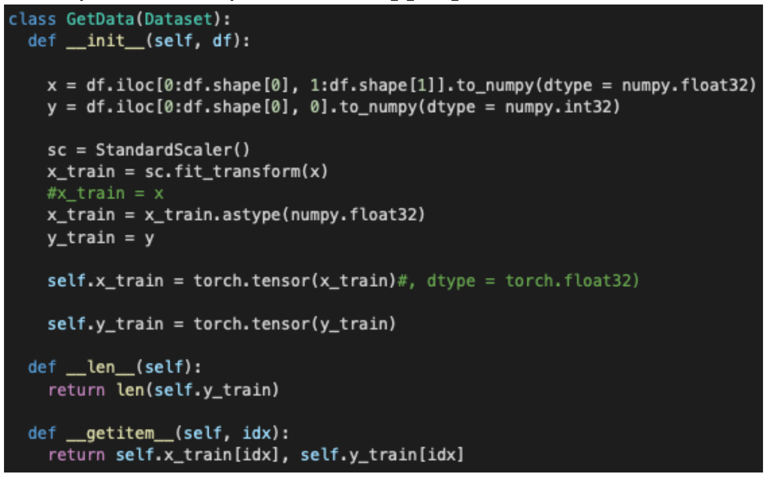
2. *Data Segregation*
   - Randomized Segregation is performed:
     - 70% for Training
     - 30% for Testing

3. *Hyperparameter Tuning*



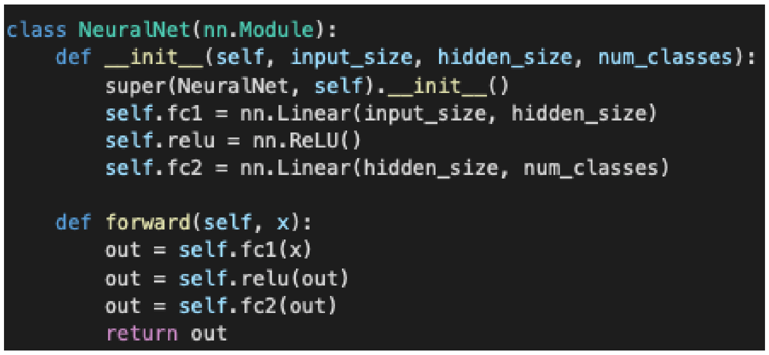
```
# Hyperparameters for the feedforward neural network

input_size = 38
hidden_size = 20
num_classes = 3
num_epochs = 20
batch_size = 1
learning_rate = 0.001

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

4. *Transformation of Data into appropriate Format:*



```
class GetData(Dataset):
  def __init__(self, df):

    x = df.iloc[0:df.shape[0], 1:df.shape[1]].to_numpy(dtype = numpy.float32)
    y = df.iloc[0:df.shape[0], 0].to_numpy(dtype = numpy.int32)

    sc = StandardScaler()
    x_train = sc.fit_transform(x)
    #x_train = x
    x_train = x_train.astype(numpy.float32)
    y_train = y

    self.x_train = torch.tensor(x_train)#, dtype = torch.float32)

    self.y_train = torch.tensor(y_train)

  def __len__(self):
    return len(self.y_train)

  def __getitem__(self, idx):
    return self.x_train[idx], self.y_train[idx]
```

5. *FeedForward Neural Network with 1 Hidden Layer*



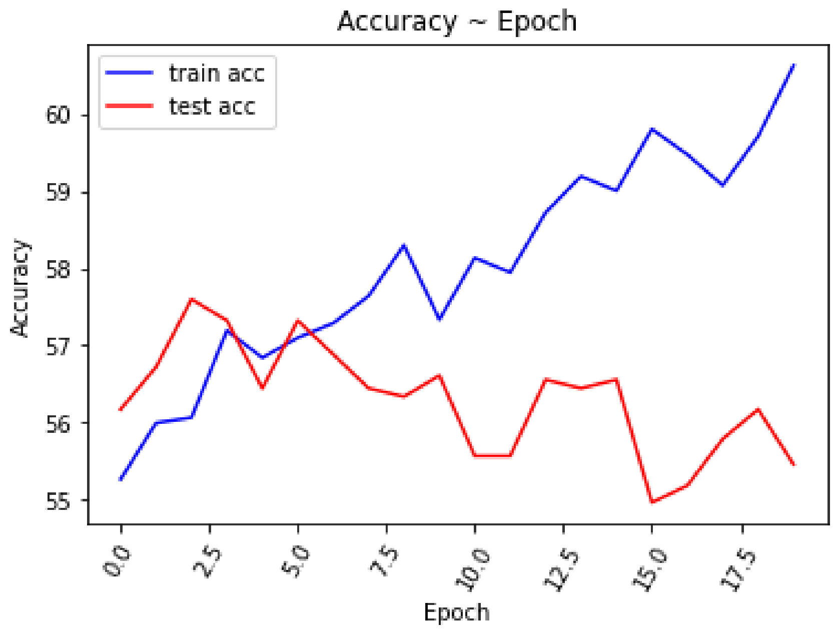
```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

- The activation function utilized – Rectified Linear Unit (ReLU)

6. *Creation of Loaders for training and testing data*

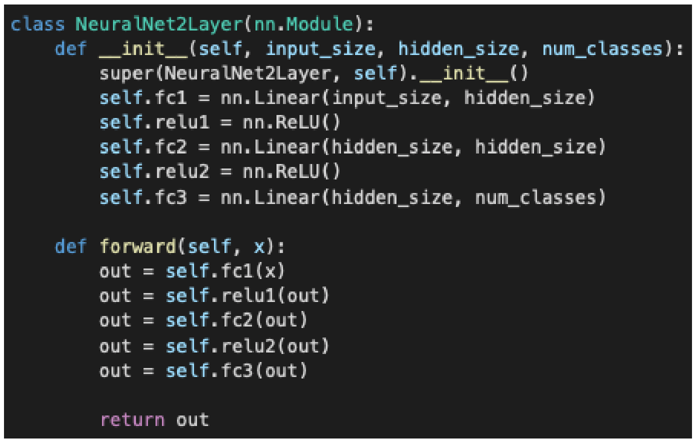
7. *Train and Accuracy Calculation*

- Results obtained are:



Accuracy obtained on the training data: 60.63%
Accuracy obtained on the testing data: 55.14%

8. *FeedForward Neural Network with 2 Hidden Layers*



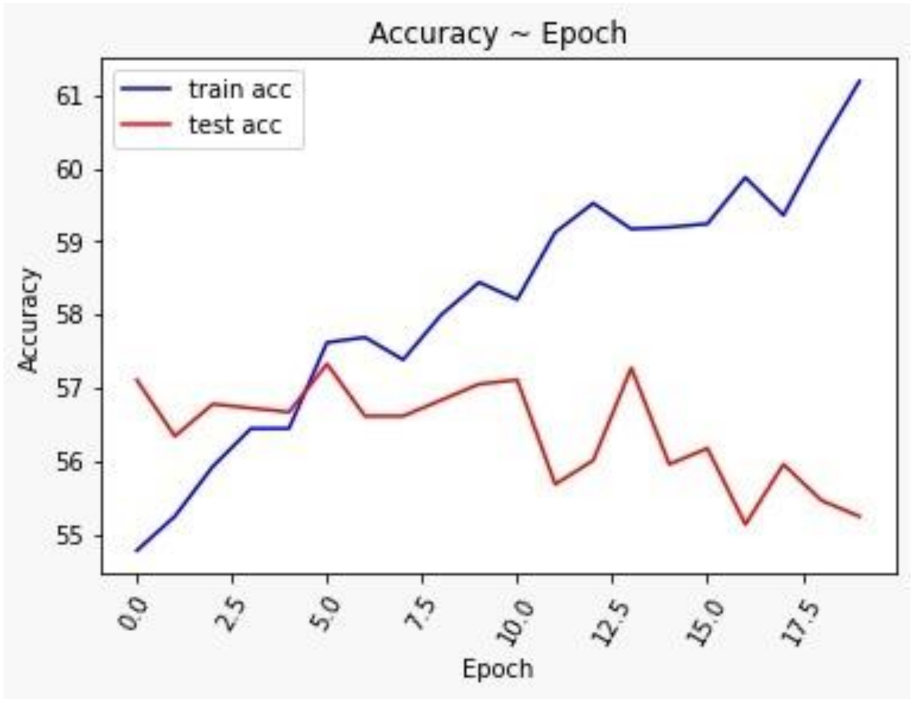
```
class NeuralNet2Layer(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet2Layer, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)

        return out
```

- The activation function utilized – Rectified Linear Unit (ReLU) (x2)

9. *Again, Creation of Loaders for training and testing data*

10. *Train and Accuracy Calculation*

- Results obtained are:



Accuracy obtained on the training data: 61.19%
Accuracy obtained on the testing data: 55.24%

Analysis: From the above two graphs, both the feed forward (FF) networks performed similar however the FF with 2 hidden layers yielded slightly better accuracy results, which makes intuitive sense as there is an additional layer to calculate the weighted probabilities in the latter.

11. *Calculation of Profit*

- The testing accuracy is run with additional properties to simulate the betting for each match in testing data.

- We used three popular betting strategies:

  a. **Flat – Betting Strategy**: Utilize the same amount for every bet. Normally - 5% of initial amount of money. Conservative and less risky if the accuracy of betting predictions is legit. But doesn't give much return.

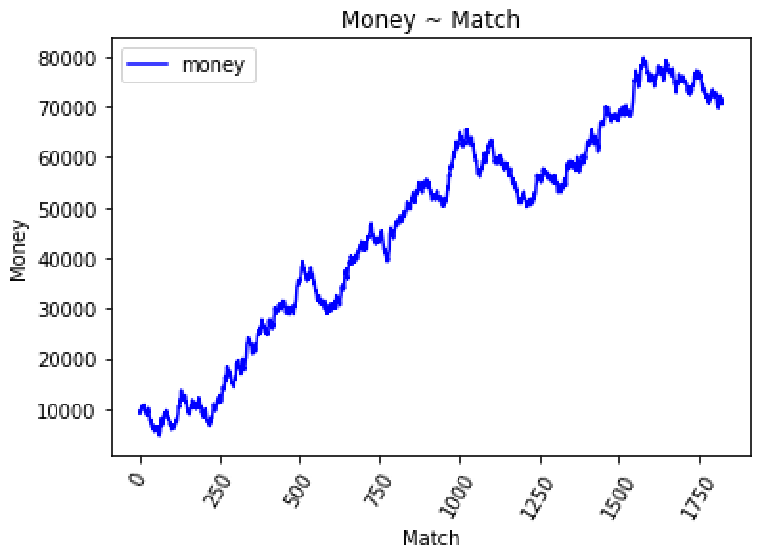
  b. **Martingale strategy**: Initial bet is 5% of the initial amount of money. If the bet is lost, double the bet (it becomes 10% of the initial amount). If we lose again - double previous one (20%) and so on until we win. Risky strategy, requires to have a lot of backup money in case of a long loss streak but is the best in terms of getting more return.

  c. **2-6 strategy**: requires winning at least 2 out of 6 bets (33%). We start with 5% of the initial bank. Then no matter what outcome, we double the bet for the next prediction. If won - we end the series and start a new one with bet equal to 5% of initial bank again. If lost - we use x4 of initial bet. If lost again - x6, then if lost once more - x8 and if this one is lost as well, x12. (max 6 bets in series). If x12 is lost, we start again from x1 of 5%.

Whenever we win twice in one series of bets (x1, x2, x4, x6, x8, x12), we end the series and start over again. 2-6 is the best strategy we know so far, since it's reliable, not much risk involved but the outcome is much higher than the Flat strategy. Also, you only need accuracy of 33% to earn money.

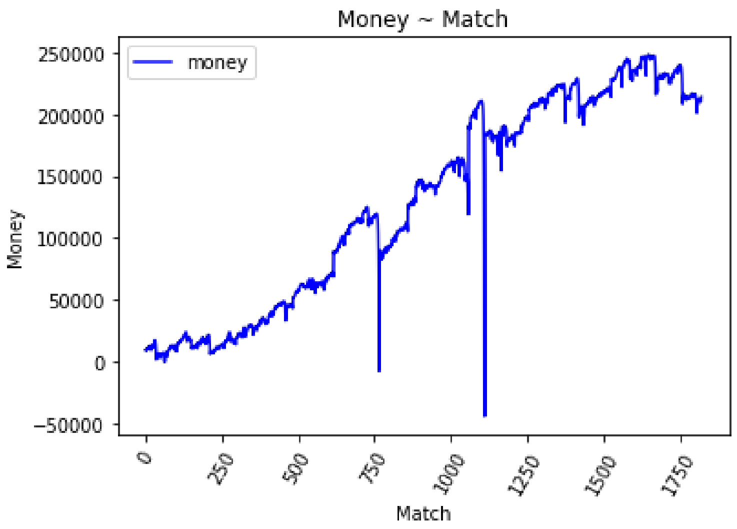
Profit Charts Obtained:

1. *Flat-Betting Strategy*



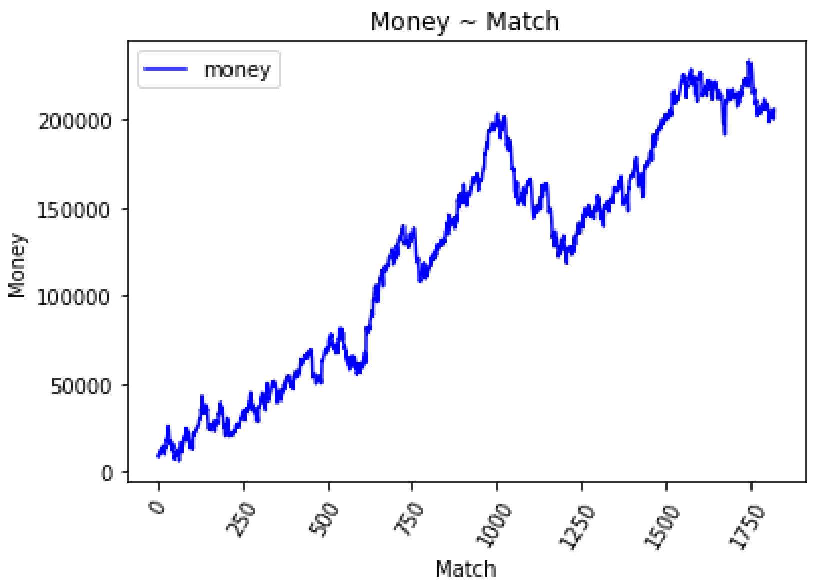
Analysis: $71,530 is the final profit return with $10,000 as initial investment. The strategy shows a stable plot of the income, but the profit is minimal. Less risky but also not as profitable strategy as other 2.

2. *Martingale Strategy*



Analysis: $214274 is the final profit return with $10000 as initial investment for Martingale strategy. It involves lots of risk and we can see by the plot that some backup amount of money is needed in case of a long loss streak (huge drops to negative values).

3. *2-6 Strategy*:



Analysis: $205,975 is the profit after the initial investment being $10,000. This strategy is the most optimal since the progression of the investment is stable, and the return rates are much higher in comparison to the flat strategy. Despite the martingale yielding the highest profit return, the 2-6 is safer since you only need to win 2 out of the 6 bets in a series.

Error Analysis:

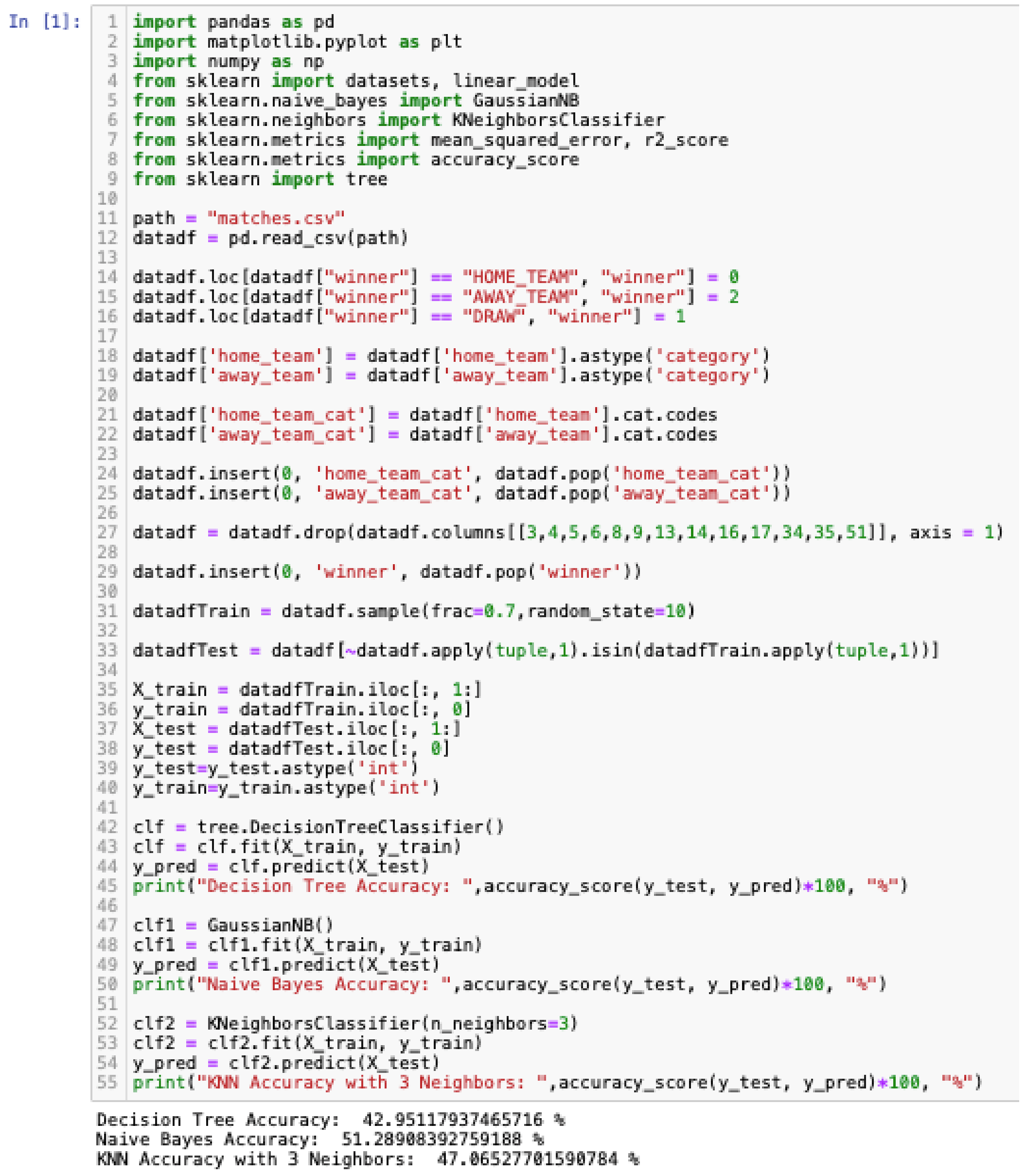
- For the Error Analysis, we instantiated 3 supervised algorithms:
  1. Decision Tree
  2. Naïve Bayes
  3. K Nearest Neighbors (KNN)

- Three supervised learning techniques were also employed to train distinct models, each of which was then compared to the neural network's performance measurements (accuracy in this case). The results are pretty pleasing, as the neural network outperforms each of the Supervised Machine Learning Algorithms tested, with an accuracy of 55%, which is more than adequate given the problem statement.

Decision Tree Accuracy (%)	Naïve Bayes Accuracy (%)	K-Nearest Neighbor (%)
42.95	51.29	47.06

- All the three accuracies obtained are less then 55% which proves to be futile for our purpose, and thus clearly indicates that Neural Networks in general have superior prediction rates that too being unsupervised in nature.

Below is a snippet of the code for the supervised algorithms used for the Error Analysis.



In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import accuracy_score
from sklearn import tree

path = "matches.csv"
datadf = pd.read_csv(path)

datadf.loc[datadf["winner"] == "HOME_TEAM", "winner"] = 0
datadf.loc[datadf["winner"] == "AWAY_TEAM", "winner"] = 2
datadf.loc[datadf["winner"] == "DRAW", "winner"] = 1

datadf['home_team'] = datadf['home_team'].astype('category')
datadf['away_team'] = datadf['away_team'].astype('category')

datadf['home_team_cat'] = datadf['home_team'].cat.codes
datadf['away_team_cat'] = datadf['away_team'].cat.codes

datadf.insert(0, 'home_team_cat', datadf.pop('home_team_cat'))
datadf.insert(0, 'away_team_cat', datadf.pop('away_team_cat'))

datadf = datadf.drop(datadf.columns[[3,4,5,6,8,9,13,14,16,17,34,35,51]], axis = 1)

datadf.insert(0, 'winner', datadf.pop('winner'))

datadfTrain = datadf.sample(frac=0.7,random_state=10)

datadfTest = datadf[~datadf.apply(tuple,1).isin(datadfTrain.apply(tuple,1))]

X_train = datadfTrain.iloc[:, 1:]
y_train = datadfTrain.iloc[:, 0]
X_test = datadfTest.iloc[:, 1:]
y_test = datadfTest.iloc[:, 0]
y_test=y_test.astype('int')
y_train=y_train.astype('int')

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Decision Tree Accuracy: ",accuracy_score(y_test, y_pred)*100, "%")

clf1 = GaussianNB()
clf1 = clf1.fit(X_train, y_train)
y_pred = clf1.predict(X_test)
print("Naive Bayes Accuracy: ",accuracy_score(y_test, y_pred)*100, "%")

clf2 = KNeighborsClassifier(n_neighbors=3)
clf2 = clf2.fit(X_train, y_train)
y_pred = clf2.predict(X_test)
print("KNN Accuracy with 3 Neighbors: ",accuracy_score(y_test, y_pred)*100, "%")
```

```
Decision Tree Accuracy:  42.95117937465716 %
Naive Bayes Accuracy:  51.28908392759188 %
KNN Accuracy with 3 Neighbors:  47.06527701590784 %
```

Future Work:

- Better correlations could be made between the features and labels and essentially figure out those features that are extremely relevant for prediction and drop those feature that hinder the accuracy.

- Implement additional neural network models to observe the prediction and profit rates.

References:

- E. Tiwari, P. Sardar and S. Jain, "Football Match Result Prediction Using Neural Networks and Deep Learning," 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2020, pp. 229-231, doi: 10.1109/ICRITO48877.2020.9197811.
- J. Hucaljuk and A. Rakipović, "Predicting football scores using machine learning techniques," 2011 Proceedings of the 34th International Convention MIPRO, 2011, pp. 1623-1627.