NBA GAME OUTCOME PREDICTION

EEE-485

PROJECT FIRST PROGRESS REPORT

Group - 3

Tugay Balatlı (21702822)

Ata Berk Çakır (21703127)

Introduction and Problem Definition

Basketball is one of the most popular sports around the world, with the use of statistics, following it is getting more and more exciting. Due to the statistical and algebraic roots of the machine learning, combining a game which creates statistically considerable data continuously with such a science can give entertaining and accurate results.

In this project, we focused on predicting NBA regular season games' outcomes. Due to the binary outcome of the NBA games, we started to work on the classification problem instances and possible solutions. Then, we decided to work on Logistic Regression, Decision Tree Classifier, Support Vector Machine Classifier and possibly K-Nearest Neighbor. Our solutions are focusing on binary classification; hence, the results will be interpreted as win (1) or lose (0).

Dataset Description

To support our models with useful data, we did some research on the topic. Initially, we tried to comprehend which statistical NBA game data is useful for us. To obtain the expert opinion and basic knowledge about the problem, we examined several articles. As Vaidya (2019) stated, "Offensive rating, defensive rating, field goal percentage, field goal attempts, 3-point percentage, 3-point attempts, assist/turnover ratio, rebound differential and pace" are important variables in our problem [1].

Our first intention was scraping our data from the basketball-reference website [2] to get the played matches and their results individually and NBA website's stats section [3] to get the statistics of the teams for the given time interval. We collected the data manually from the given websites. Then, the collected data instances from 2016-2017, 2017-2018, 2018-2019 and 2020-2021 regular seasons. The reason why we excluded 2019-2020 regular season is because of the pandemic. With the light of the common knowledge and the article written by Botkin and Blomain, we observed that there are a few data instances in this regular season [4]. Additionally, since the league is suspended for a long time, the real statistics were not totally reflected in the dataset regarding 2019-2020. After the collection process, we merged our data with respect to the index of the games as all the games are scheduled in an order.

In our data, we have 3370 instances and 32 columns. Half of the columns include statistics and name of the home team. Other half contains same statistics for the away team for the same game. The statistics are numeric values about the points made in the given game ("PTS") , offence, defense and net ratings ("Team1OFFRTG", "Team1DEFRTG", "Team1NETRTG"), assist, rebound (offensive and defensive), assist to turnover ratio ("Team1AST%", "Team1AST/TO", "Team1OREB%", "Team1DREB%", "Team1AST RATIO"), turnover ratio ("Team1TOV%"), true shooting percentage ("Team1TS%"). In addition to these columns, we have two additional columns that are calculated by the NBA statisticians with these methods: Pace: 48 * ((Team Possessions + Opponent Possessions) / (2 * (Team Minutes Played / 5))) [5], PIE: % of game events did that player or team achieve [6] (Team1PACE, Team1PIE). These columns are included in the opponent's statistics.

After completed the data inspection and collection part, we did some preprocessing on our dataset. First of all, we created a to assign response variable which is called "Team1Win". It has binary values in it and refers to the home team's win situation.

In the preprocessing part, we created a function to apply standardization by standard scaling using the formula given below:

$$z = x^{scaled} = \frac{x - \mu}{\sigma}$$

Hence, we obtained scaled data to decrease the effect of the outliers in numeric columns. Then, we dropped points made in the game, team names from the columns to avoid memorization and hence overfitting.

In the last step of the data preprocessing, predictor and response variables are separated into two sets named as X for predictors and y for response. After creating these two data frames, we split our dataset into train and test set by coding a function specifically for this purpose. This function takes split size as

an input. As a result of our literature review, we found out that using 0.8 would be efficient for the model training and testing parts.

Lastly, the multicollinearity is considerable problem for the performance of the models. Therefore, we controlled the correlations between the features (Can be seen in the figure below). After some inspection, we observed that there is a high correlation between columns "Team1AST RATIO" and "Team1AST%". The same observation is also acceptable for the Team 2. Hence, we decided to drop "Team1AST RATIO". Similar observations are done for the "Team1NETRTG" and "Team1PIE", "Team2EFG%" and "Team2TS%" duos. Thus, we also decided to drop "Team1PIE" and "Team2TS%" for both teams. We used Seaborn and Matplotlib libraries to visualize our correlation results.

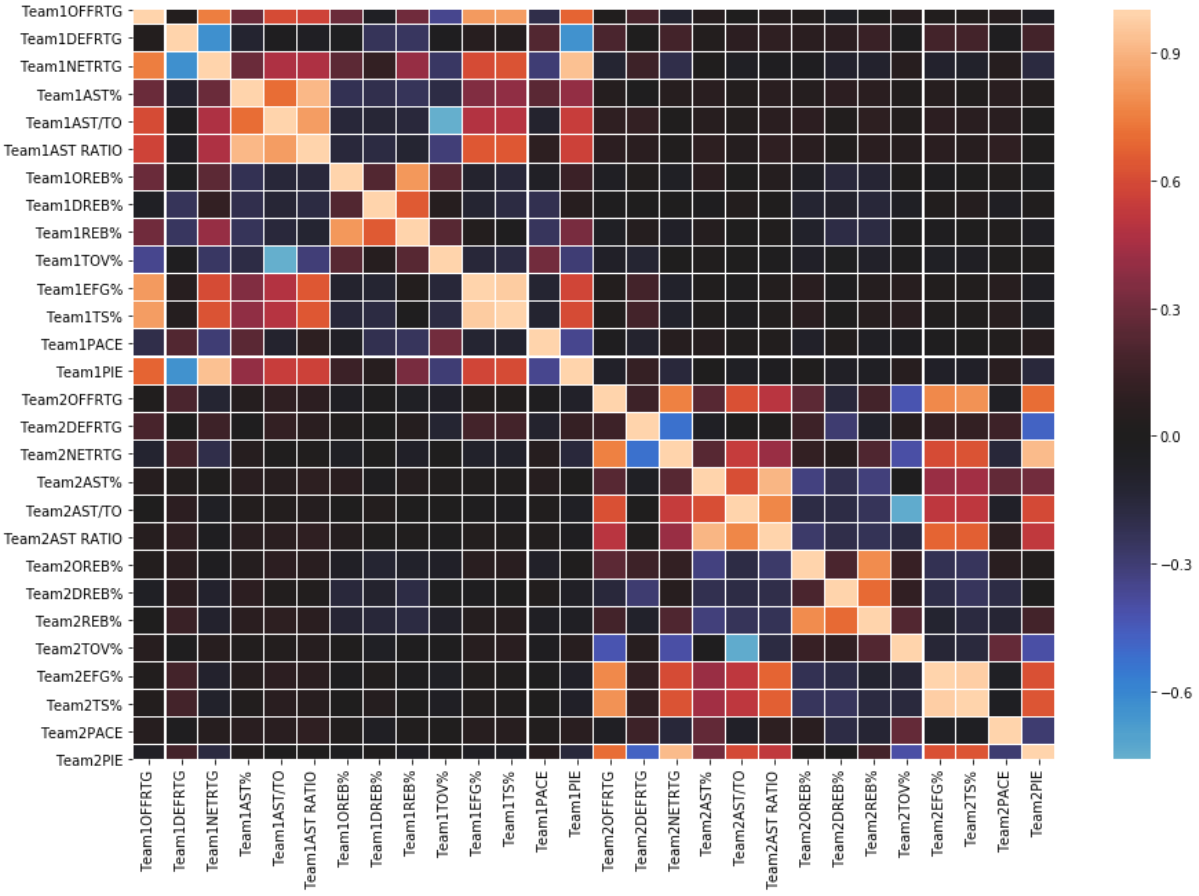

Figure 1 – Correlation Matrix as a Heat Map

Methods

The algorithms that are selected for the purpose of the project are Logistic Regression, Support Vector Machine and Decision Tree. We aimed to get binary results as it is stated above, therefore, using these algorithms can give desired outcome.

Method 1 – Logistic Regression

When the predicted outcome of the model is categorical, logistic regression can be used. Since we are trying to predict the game results which 1 or 0 the logistic regression model fits best to our problem therefore, we decided to use this method. We are trying to learn directly $P(Y|X)$ with assuming $P(Y|X)$ takes functional form. In other words, sigmoid function applied to a linear function of the data. The sigmoid funtion is shown in the following figure.

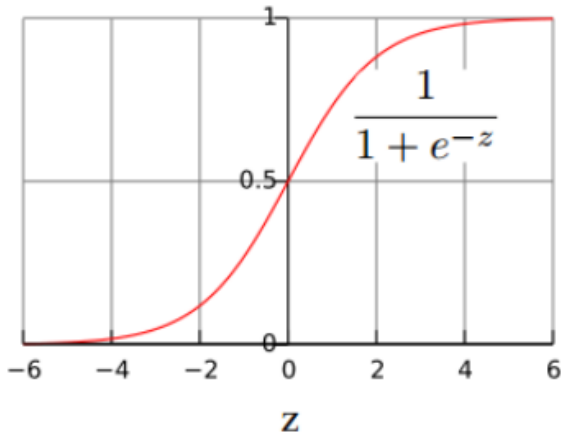

Figure 2 – Logistic Function (Sigmoid Function)

The calculated probability is the hypothesis's output. When given an input X, this is utilized to determine how certain a predicted value can be the actual value. The probabilities are calculated by the following formulas [8].

$$P(Y=1|X) = \frac{1}{1+e^{(w_0 \sum_{i=1}^{n} w_i X_i)}} \quad (1)$$

$$P(Y=0|X) = \frac{e^{(w_0 \sum_{i=1}^{n} w_i X_i)}}{1+e^{(w_0 \sum_{i=1}^{n} w_i X_i)}} \quad (2)$$

The predicted value for Y will be equal to the output of the class with highest $P(Y|X)$.

$$1 < \frac{P(Y=0|X)}{P(Y=1|X)} \quad (3)$$

$$1 < e^{(w_0 \sum_{i=1}^{n} w_i X_i)} \quad (4)$$

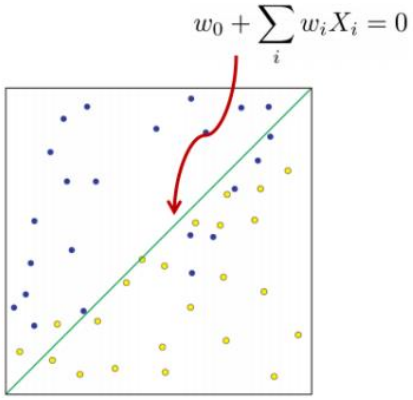

Figure 3 – Linear Decision Boundary

In order to learn parameters $w_0, w_1, \ldots \ldots, w_n$ we are going to use maximum conditional likelihood estimator which is the following where training data is $\{X^{(j)}, Y^{(j)}\}_{j=1}^{n}$

$$\widehat{w}_{MCLE} = arg\max_{w} \prod_{j=1}^{n} P\left(Y^{(j)} \middle| X^{(j)}, w\right) \quad (5)$$

Since conditional likelihood for Logistic Regression is concave, we are going to use Gradient Ascent to optimize parameters.

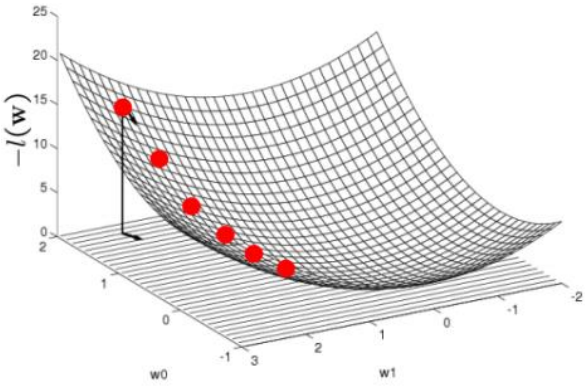

Figure 4 – Gradient Ascent

Gradient:

$$\nabla_w l(w) = \left[\frac{\partial l(w)}{\partial w_0}, \dots\dots, \frac{\partial l(w)}{\partial w_n}\right] \tag{6}$$

Update rule (where n is learning rate):

$$\Delta w = n\nabla_w l(w) \tag{7}$$

Method 2 – Support Vector Machine Classifier

We are going to use SVM classifier for linearly separable classes for problem. An SVM training algorithm creates a model that assigns new examples to one of two categories, making it a non-probabilistic binary linear classifier, given a series of training examples, each marked as belonging to one of two categories [7]. The key idea behind SVM is hyperplane that maximizes the margin will have better generalization. For a linearly separable data SVM can better be understand from the following figure.

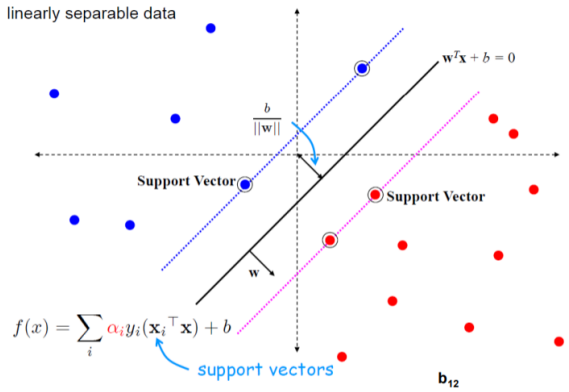

Figure 5- Support Vector Machine

Since $w^T x + b = 0$ and $c(w^T x + b) = 0$ define the same plane we can decide the normnalization of w such that $w^T x_+ + b = +1$ and $w^T x_- + b = -1$ for the positive and negative support vectros respectively.

The margin can be found by

$$\frac{w}{\|w\|} * (x_+ - x_-) = \frac{w^T(x_+ - x_-)}{\|w\|} = \frac{2}{\|w\|}$$

The geometric representation of SVM is the following

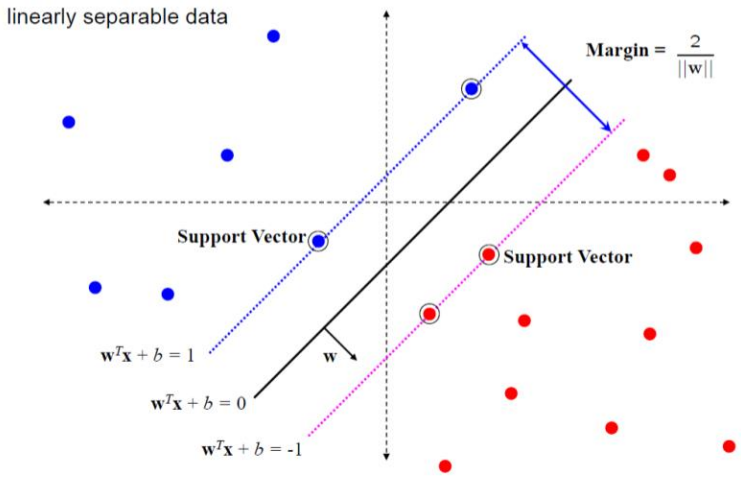

Figure 6 – Geometric representation of SVM

Then learning SVM can be formulated as optimization problem where it is a quadratic optimization problem subject to linear constraints and there exist a unique minimum [8].

$$\min_{w} \|w\|^2 \; subject\ to\ y_i(w^T x_i + b\,) \geq 1\ for\ i = 1 \ldots \ldots N$$

Method 3 – k-Nearest Neighbor Classifier

In the k-NN classifier we are assuming all instances are points in n-dimensional space and we are required to use a distance measure to determine the "closeness" of instances. Although there are different types of distance metrics such as Minkowski distance, Manhattan distance, Cosine distance, Jaccard distance; Euclidean distance fits bets for our case. Given all that NBA performance measurements are numeric and have the same units, we can directly use Euclidean distance defined as the square root of the sum of the squared differences between the two arrays of number. In our case each row of the future set can be considered as an array of number. Finally, we are classifying an instance by finding its nearest neighbors and picking the most popular class among the neighbors [8].

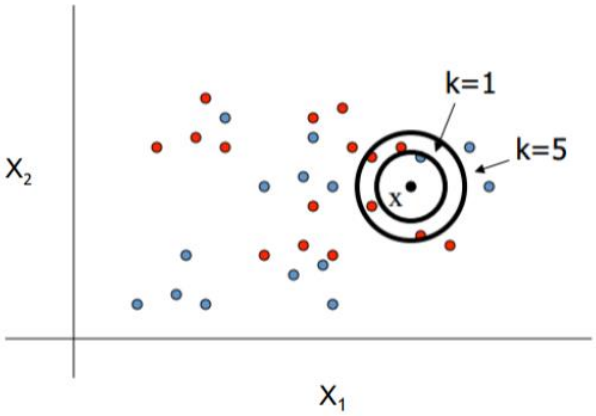

Figure 7 – k-NN Classifier

Method 4 – Decision Tree Classifier

In this method, we aimed to use decision tree classifier. It is based on a tree structure and decisions made on the internal nodes and branches. As a result, the leaves show the respective class for the selected specific part of the tree. The separation in the branches to classes is works with the if statements in general. The classification and node structures are based on several concepts in decision tree classifier. Those are entropy, information gain and Gini index.

As a stopping criterion for splitting, the algorithm uses the purity of the child node. It is done by checking entropy and information gain regarding the entropy.

Information is reduction in the uncertainty in outcome. Its formula is:

$$I(X) = -log_2 p(x)$$

Entropy is expected amount of the information while observing the output of random variable X. Formula for the entropy is:

$$H(X) = E(I(X)) = \Sigma_i p(x_i) I(x_i) = -\Sigma p(x_i) log_2 p(x_i)$$

Information gain is reduction in uncertainty by knowing Y. The formula for information gain is:

$$IG(X,Y) = H(Y) - H(Y|X)$$

Information gain also helps us to indicate which attributes in the data set's feature column are helpful for the discrimination of the given classes. Information gain value shows us the importance as a metric.

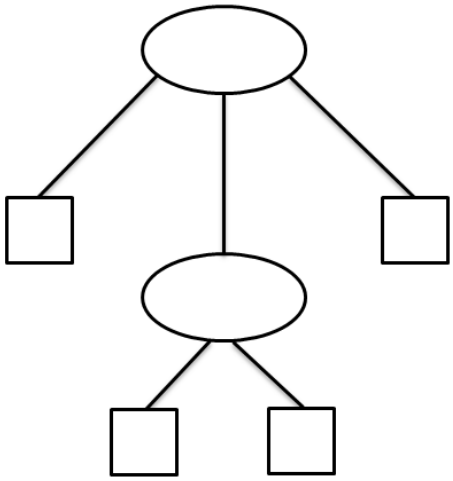

Figure 8 – Decision Tree Classifier Example

As it is stated above, the purity is the decision mechanism for the stopping conditions. Gini index is calculated to understand the purity. Where $p(i|t)$ is the relative frequency of class i at node t. Here is the formula of Gini index:

$$GINI(t) = 1 - \Sigma_i p(i|t)^2$$

Lower Gini index values are considered for the further investigation which is done by adding new branches. [8]

Completed Milestones

Our initial focus was creating a neat and functional dataset. Before merging the datasets, we created a Python code to assign binary values based on comparison of points made by each team, and home team's win or lose situation (See the Appendix for the code implementations). It has simple logic behind it similar to binary encoders, such as One Hot Encoder. As we do not have any other categorical value, we will not face with a challenge about encoding.

We expected to see some challenges. First of all, outlier values were our first concern, however, after applying standardization by using Standard Scaler algorithm, we fixed the possible problems in general. As we are focusing on the binary results, we will not need to rescale the results. The new data that can be added should be revised before prediction or train part because of the scaling process. Additionally, the class imbalance was not a big problem for us as there is always one winner and one loser.

As we planned, we created train and test sets by homogeneously splitting our data set. Data cleaning and preprocessing is completed. We focused on avoiding multicollinearity, which indicates that there is strong inter-correlation between two or more independent variable, to increase our models' performance.

We started to implement the algorithms after the data preprocessing part. We are currently implemented k-nearest neighbor algorithm and we are working on the logistic regression and decision trees for now. We are going to test our models with different metrics. These can be recall, precision and accuracy values, but the accuracy is not known as strong metric to indicate model's performance. Hence, we are going to try to widen our performance by checking other metrics. We are going to use confusion matrix to assess these metrics in numerical values.

After this point, we will widen the scope of our progress by implementing the support vector machine classifier. Then, we can work on the model enhancements by shifting the input parameters of the models, i.e. changing the threshold probability value for the logistic regression or maybe by working on the regularization methods, such as L1 and L2 regularizations.

In the further steps, we are planning to follow the roadmap and schedule that we created below in the last quarter of 2021. Each group member worked and will work on the all work packages of the project stated below.

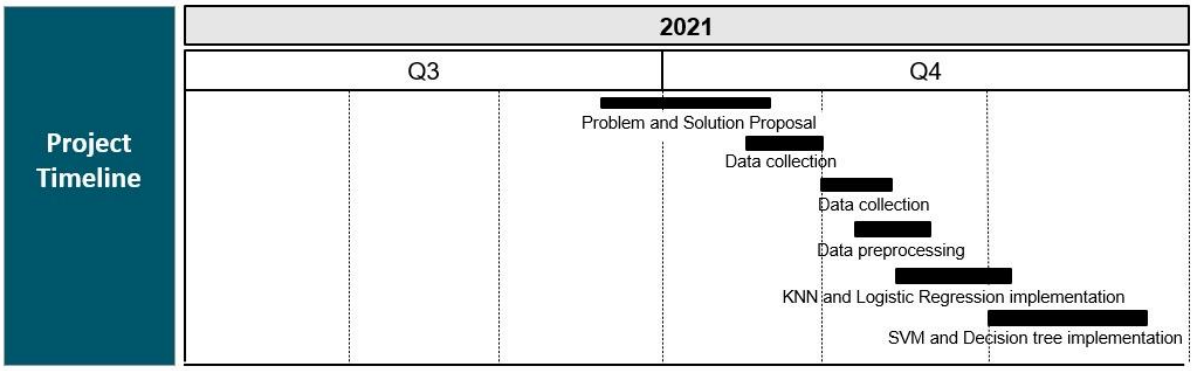

Figure 9 – Gantt Chart of The Project Plan

Challenges

The implementation of the support vector machine classifier will be our first challenge. After the implementations are done, we are going to focus on our codes' performance in terms of computation time. For now, our computation time in k-NN algorithm is around 30 seconds to 1 minute which is quite good. But the other models will be considered in the scope of this challenge. At the end, the parameter optimization and perhaps the grid search algorithm will be our last and biggest challenge about the model performance.

References

[1] C. Vaidya. (2019). Which NBA Statistics Actually Translate to Wins? [Online]. Available: https://watchstadium.com/which-nba-statistics-actually-translate-to-wins-07-13-2019/ [Accessed: 17-Nov-2021].

[2] Basketball-Reference. (2021). NBA Schedule and Results [Online]. Available: https://www.basketball-reference.com/leagues/NBA_2021_games.html [Accessed: 17-Nov-2021].

[3] NBA. (2021). NBA Advanced Stats [Online]. Available: www.nba.com/stats/teams [Accessed: 17-Nov-2021].

[4] B. Bodkin & M. Kaskey-Blomain. (2020). NBA Suspends Season Due to Coronavirus Outbreak; Owners Preparing For No Games Until June, Per Report [Online]. Available: https://www.cbssports.com/nba/news/nba-suspends-season-due-to-coronavirus-outbreak-owners-preparing-for-no-games-until-june-per-report/ [Accessed: 17-Nov-2021].

[5] Basketball-Reference. (2021). Glossary [Online]. Available: https://www.basketball-reference.com/about/glossary.html#mp [Accessed: 18-Nov-2021].

[6] NBA (2021). Frequently Asked Questions – What is PIE [Online]. Available: https://www.nba.com/stats/help/faq/#:~:text=In%20its%20simplest%20terms%2C%20PIE,to%20be%20a%20winning%20team [Accessed: 18-Nov-2021].

[7] C. Hsu & C. Chang & C. Lin (2016). A Practical Guide to Support Vector Classification [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf [Accessed: 21-Nov-2021].

[8] G. James, D. Witten, T. Hastie, R. Tibshirani (2013). An Introduction to Statistical Learning. Springer. p. 6.

Appendix

The following codes can be found in the .py files. Appendix – 1 can be found in NBA_feature_extraction.py. Appendix – 2, 3, 4, 5 ,7 can be found in Logistic_Regression.py. Lastly, Appendix – 6 can be found in kNN.py.

Appendix – 1: Data Preprocessing & Feature Extraction Codes

```
import pandas as pd
import numpy as np
df = pd.DataFrame()
for i in range(1,6):
   stats = pd.read_excel(open("C:/Users/ASUS PC/Desktop/NBA Data/16-17/{}.xlsx".format(i),
'rb'),sheet_name = "Sheet1")
   results = pd.read_excel(open("C:/Users/ASUS PC/Desktop/NBA Data/16-17/{}.xlsx".format(i),
'rb'),sheet_name = "Sheet2")
   # Adding a new binary columns for Team 1 Win or Loss
   conditions = [
      results['PTS'] > results['PTS.1'],
      results['PTS'] < results['PTS.1']
   ]
   choices=[1,0]
   results['Team1Win'] = np.select(conditions, choices, 1)
   # Mergings the results and stats datasets, adding a prefix depending on which teams stats are being
used
   df1 = results.merge(stats.add_prefix('Team1'), how='left', left_on=['Visitor/Neutral'],
                right_on=['Team1TEAM']).drop(['Team1TEAM'],
                                    axis=1).merge(stats.add_prefix('Team2'),
                                                  how='left', left_on=['Home/Neutral'],
                                                  right_on=['Team2TEAM']).drop(['Team2TEAM'],ax
is=1)
df = pd.concat([df,df1])
   #Visitor/Neutral = Team1
for i in range(1,6):
   stats = pd.read_excel(open("C:/Users/ASUS PC/Desktop/NBA Data/17-18/{}.xlsx".format(i),
'rb'),sheet_name = "Sheet1")
   results = pd.read_excel(open("C:/Users/ASUS PC/Desktop/NBA Data/17-18/{}.xlsx".format(i),
'rb'),sheet_name = "Sheet2")
   # Adding a new binary columns for Team 1 Win or Loss
   conditions = [
      results['PTS'] > results['PTS.1'],
      results['PTS'] < results['PTS.1']
   ]
   choices=[1,0]
   results['Team1Win'] = np.select(conditions, choices, 1)
   # Mergings the results and stats datasets, adding a prefix depending on which teams stats are being
used
   df2 = results.merge(stats.add_prefix('Team1'), how='left', left_on=['Visitor/Neutral'],
                right_on=['Team1TEAM']).drop(['Team1TEAM'],
```

```
                                    axis=1).merge(stats.add_prefix('Team2'),
                                                  how='left', left_on=['Home/Neutral'],
                                                  right_on=['Team2TEAM']).drop(['Team2TEAM'],axis=1)
df = pd.concat([df,df1])
    #Visitor/Neutral = Team1
for i in range(1,6):
    stats = pd.read_excel(open("C:/Users/ASUS PC/Desktop/NBA Data/18-19/{}.xlsx".format(i),
'rb'),sheet_name = "Sheet1")
    results = pd.read_excel(open("C:/Users/ASUS PC/Desktop/NBA Data/18-19/{}.xlsx".format(i),
'rb'),sheet_name = "Sheet2")
    # Adding a new binary columns for Team 1 Win or Loss
    conditions = [
        results['PTS'] > results['PTS.1'],
        results['PTS'] < results['PTS.1']
    ]
    choices=[1,0]
    results['Team1Win'] = np.select(conditions, choices, 1)
    # Mergings the results and stats datasets, adding a prefix depending on which teams stats are being
used
    df3 = results.merge(stats.add_prefix('Team1'), how='left', left_on=['Visitor/Neutral'],
                right_on=['Team1TEAM']).drop(['Team1TEAM'],
                                    axis=1).merge(stats.add_prefix('Team2'),
                                                  how='left', left_on=['Home/Neutral'],
                                                  right_on=['Team2TEAM']).drop(['Team2TEAM'],axis=1)
    df = pd.concat([df,df1])
    #Visitor/Neutral = Team1
for i in range(1,6):
    stats = pd.read_excel(open("C:/Users/ASUS PC/Desktop/NBA Data/20-21/{}.xlsx".format(i),
'rb'),sheet_name = "Sheet1")
    results = pd.read_excel(open("C:/Users/ASUS PC/Desktop/NBA Data/20-21/{}.xlsx".format(i),
'rb'),sheet_name = "Sheet2")
    # Adding a new binary columns for Team 1 Win or Loss
    conditions = [
        results['PTS'] > results['PTS.1'],
        results['PTS'] < results['PTS.1']
    ]
    choices=[1,0]
    results['Team1Win'] = np.select(conditions, choices, 1)
    # Mergings the results and stats datasets, adding a prefix depending on which teams stats are being
used
    df4 = results.merge(stats.add_prefix('Team1'), how='left', left_on=['Visitor/Neutral'],
                right_on=['Team1TEAM']).drop(['Team1TEAM'],
                                    axis=1).merge(stats.add_prefix('Team2'),
                                                  how='left', left_on=['Home/Neutral'],
                                                  right_on=['Team2TEAM']).drop(['Team2TEAM'],axis=1)

    df = pd.concat([df,df1])
    #Visitor/Neutral = Team1
```

```
df=df.drop(["Team1GP","Team1W","Team1L","Team1MIN","Team1POSS","Team2GP","Team2W"
,"Team2L","Team2MIN","Team2POSS"], axis = 1)
df.to_excel(r'C:/Users/ASUS PC/Desktop/raw_data.xlsx', index = False, header=True)
print(df.info())
statistics = df.describe().T
```

Appendix – 2: Data Cleaning & Preprocessing Codes

```
raw_data = pd.read_excel("C:/Users/ASUS PC/Desktop/4.2/EEE 485/EEE485
Project/Data/raw_data.xlsx")
scale_columns = ['Team1OFFRTG', 'Team1DEFRTG', 'Team1NETRTG',
                 'Team1AST%', 'Team1AST/TO', 'Team1OREB%',
                 'Team1DREB%', 'Team1REB%', 'Team1TOV%', 'Team1EFG%', 'Team1TS%',
                 'Team1PACE', 'Team2OFFRTG', 'Team2DEFRTG', 'Team2NETRTG',
                 'Team2AST%', 'Team2AST/TO', 'Team2OREB%',
                 'Team2DREB%', 'Team2REB%', 'Team2TOV%', 'Team2EFG%', 'Team2TS%',
                 'Team2PACE']
for i in scale_columns:
    StandardScaler(raw_data, i)
#Splitting data to predictors and response
y = raw_data["Team1Win"]
X = raw_data.drop(["Visitor/Neutral","PTS","Home/Neutral","PTS.1","Team1Win","Team1AST
RATIO", "Team2AST RATIO", "Team1PIE", "Team2PIE"], axis = 1)
```

Appendix – 3: Standard Scaler Codes

```
def StandardScaler(data, column):
    mean = np.mean(data[column])
    std = np.std(data[column])
    for i in range(data.shape[0]):
        data[column][i] = (data[column][i] - mean) / std
    return data
```

Appendix – 4: Train Test Split Codes

```
def TrainTestSplit(X, y, split = 0.8):
    X_train = X.loc[:int(raw_data.shape[0]*0.8),:].values
    X_test = X.loc[int(raw_data.shape[0]*0.8):,:].values
    y_train= y.loc[:int(raw_data.shape[0]*0.8),:].values.reshape(-1,1)
    y_test = y.loc[int(raw_data.shape[0]*0.8):,:].values.reshape(-1,1)
    return X_train, X_test, y_train, y_test
```

Appendix – 5: Correlation and Data Exploration Codes

```
import seaborn as sns
import matplotlib.pyplot as plt
raw_data = pd.read_excel("C:/Users/ASUS PC/Desktop/4.2/EEE 485/EEE485
Project/Data/raw_data.xlsx")
raw_data = raw_data.iloc[:,5:]
correlation = raw_data.corr(method='pearson')
f, ax = plt.subplots(figsize=(15, 10))
ax = sns.heatmap(correlation, center = 0,  linewidths=.1)
t = sns.pairplot(correlation)
```

Appendix – 6: k-Nearest Neighbor Codes

```
import pandas as pd
import math
from timeit import default_timer as timer

start = timer()
def TrainTestSplit(X, y, split = 0.8):
    X_train = X[:int(raw_data.shape[0]*0.8)]
    X_test = X[int(raw_data.shape[0]*0.8):]
    y_train= y[:int(raw_data.shape[0]*0.8)]
    y_test = y[int(raw_data.shape[0]*0.8):]
    return X_train, X_test, y_train, y_test

raw_data = pd.read_excel("raw_data.xlsx")

#Splitting data to predictors and response
y = raw_data["Team1Win"]
X = raw_data.loc[:, raw_data.columns != "Team1Win"]

X_train, X_test, y_train, y_test = TrainTestSplit(X,y)

y_test = y_test.to_frame()
y_train = y_train.to_frame()

y_test['index_col'] = y_test.index
y_train['index_col'] = y_train.index
X_train['index_col'] = X_train.index
X_test['index_col'] = X_test.index

df_train = pd.merge(X_train, y_train,how="left")
df_test = pd.merge(X_test, y_test,how="left")
df_train = df_train.iloc[:,5:]
df_test = df_test.iloc[:,5:]
df_test = df_test.drop('index_col',1).values
df_train = df_train.drop('index_col',1).values
```

```
y_test = y_test.drop('index_col',1).values
y_train = y_train.drop('index_col',1).values


def get_euclidean_distance (train_row,test_row):
    dist = 0.0
    for i in range(len(train_row)-1):
        dist += (train_row[i]-test_row[i])**2
    return math.sqrt(dist)

#gets the k nearest neighbors of a single row of the test data
def get_neighbors (train,test_row,k):
    distances = []
    neighbors =[]
    for train_row in train:
        eu_dist = get_euclidean_distance(train_row,test_row)
        distances.append((train_row,eu_dist))
    distances.sort(key = lambda tup: tup[1])
    for i in range(k):
        neighbors.append(distances[i][0])
    return neighbors

#makes prediction based on a single row of the test data
def predict_classification(train, test, k):
    predictions = []
    for test_row in test:
        outcome = []
        neighbors = get_neighbors(train, test_row, k)
        for row in neighbors:
            outcome.append(row[-1])
        prediction = max(outcome, key=outcome.count)
        predictions.append(prediction)
    return predictions

def calculate_accuracy(predicted_labels,test_labels):
    tp = 0.0 #true positive count
    tn = 0.0 #true negative count
    fp = 0.0 #false positive count
    fn = 0.0 #false negative count
    for i in range(len(predicted_labels)):
        prediction = predicted_labels[i]
        actual = test_labels[i]
        if ((prediction == 1) & (actual == 1)):
            tp+=1
        elif ((prediction == 1) & (actual == 0)):
            fp+=1
        elif ((prediction == 0) & (actual == 0)):
            tn+=1
        elif ((prediction == 0) & (actual == 1)):
```

```
            fn+=1
    accuracy = ((tp+tn)/(tp+tn+fp+fn))*100
    return accuracy


predicted_labels = predict_classification(df_train,df_test, 15)
accuracy = calculate_accuracy(predicted_labels,y_test)
end = timer()

print ("Accuracy: ",accuracy)
print("time elapsed: ",end - start)
```

Appendix – 7: Logistic Regression Codes (Not Completed Yet)

```
class LogisticRegression():

    def __init__(self, learning_rate, iterations):
        #Initializing hyperparameters for the gradient descent: learning rate and number of iterations.
        self.learning_rate = learning_rate
        self.iterations = iterations


    def fit(self, X, y):
        self.m, self.n = X.shape

        #Trains the logistic regression model for the given dataset
        #weight
        self.weight = np.zeros((self.n, 1))
        #bias
        self.bias = 0

        for i in range(self.iterations + 1):
            self.update_weights(X, y)
        return self.weight, self.bias


    def update_weights(self, X, y):
        #Sigmoid Function
        y_hat = 1 / (1 + np.exp( - X.dot(self.weight) + self.bias ))

        #To apply gradient descent, we took derivative
        dw = (1 / self.m) * np.dot(X.T, (y_hat - y))
        db = (1 / self.m) * np.sum(y_hat - y)

        #Loss function
        #self.loss = -1/self.m * np.sum(y * np.log(self.y_hat) + (1 - self.y) * np.log(1 - self.y_hat))
```

```
    #Update weight and bias
    self.weight = self.weight - self.learning_rate * dw
    self.bias = self.bias - self.learning_rate * db


def predict(self):
    y_pred = 1 / (1 + np.exp( - X.dot(self.weight) + self.bias ))
    y_pred = np.where( y_pred > 0.5, 1, 0 )
    return y_pred

def get_params(self):
    return self.w
```