Machine Learning Homework Four

MATT FERGUSON

Problem Statement

Given a population of baseball statistics and salaries for players and years:

- Develop classification models using K nearest neighbor (KNN) classification to predict the league and team of baseball players for a year.
- Develop multivariate linear regression models to predict salary of baseball players for each year in the data set and create a model using every year.

Requirements

This file contains:

- KNN classifier code that predicts team and league
- Multivariate linear regression code that predicts salary for each year and overall
- Graph of classification accuracy versus K for team and league prediction models
- Graph of mean Square Error (MSE) versus year for salary prediction model
- Discussion of preprocessing, code, graphs, and results.

Data Preprocessing

I began my assignment by inspecting the events, features, and targets in my data set. We start with a set of 45174 events. Each event is a set of statistics for a player each year. This is a key pair that creates our identifier which we will call player-year or year-player. Note that we have two sets of numeric features, features that are counts of player actions (batting attempts, runs, home runs, etc.) and rates of player actions (run rate, home run rate, etc.)

Rates are always calculated as the action divided by the number of batting attempts. For incidents where the player never bats (let's say a pitcher), this will certainly skew our statistics. Players with zero bats may be highly paid pitchers! In fact, a total of $9,096,721,024 was paid to 6414 player-years with zero at bats ($1,418,260.22 average). We must remove these records where players never made a batting attempt because they will significantly depress our correlations against salary and a count of zero at bats forms a rate that is N/A (because we would divide by zero bat attempts). Additionally, we remove all values of salary that are equal to zero, 99999.

On the right we can see my python script which loads the original batting salaries spreadsheet into excel, drops the N/A values across all rows, and then drops all rows where the batting attempts equals 0.

The data is saved as our preprocessed excel file to we can load this preprocessed file in the future to save on compute time.

After accomplishing preprocessing, we are left with 11240 events with a salary of N/A. These N/A salaries must be deleted for our regression model as salary is the target. Imputation is not an attractive option to perform on our target variable as it would bias our target towards whatever features we use for imputation. We take the above actions using python, and then load the preprocessed data into python by means of a pandas data frame. After data cleansing in excel we are left with 21704 records and 667 records for 2016 specifically. From there we will be able to proceed with modelling using the year 2016 for classification and iterate through weights of K to predict team and league. For our linear regression model, we will iterate by year through our data frame in order to determine which year has the lowest MSE when predicting salary.

```
import pandas as pd

filename = (r'C:\Users\Matt\Desktop\BattingSalaries.xlsx')
df= pd.read_excel(filename)

df=df.dropna()
df.loc[(df[['AB']] != 0).all(axis=1)]

df.to_excel(r'C:\Users\Matt\Desktop\BattingSalaries_preprocessed.xlsx')
```

KNN Classification Code

The code I developed reads the batting salaries preprocessed data into a pandas data frame and keeps only records with year-ids of 2016. We then normalize the data frame, convert features and targets to a NumPy array and then pass these to a sklearn KNN classifier for training. We iterate through k = 1 to k=15 as well as uniform and distance weightings. We passed both the team and league ID as targets to fit separate models. We output the resulting classification accuracy and parameters of these models for our iterations of K/weight to the console and have recorded/graphed these results in the following slides. To guide feature selection, we created a correlation matrix using our DQR script and rates that correlated best to the target were tested for classification accuracy.

```
import pandas as pd
import numpy as np
import sklearn as sk

filename = (r'C:\Users\Matt\Desktop\BattingSalaries_preprocessed.xlsx')
df= pd.read_excel(filename)

df=df[df['yearID']==2016]
train_target=df['teamID'].tolist()

df=df.drop(['playerID','teamID','lgID','yearPlayer'], axis=1)
df=(df-df.min())/(df.max()-df.min()) #normalize

feature_names= ['G','AB','R','H','SH','2B','3B','HR','RBI','IBBrat','Hrat']
train_features=df[feature_names].values.tolist()


X = np.array(train_features)
Y = np.array(train_target)

(trainX, testX, trainY, testY) = sk.model_selection.train_test_split(X, Y, test_size=0.3, random_state=1)

for k in range(1,16):
    for weighting in ['uniform', 'distance']:

        salary_model = sk.neighbors.KNeighborsClassifier(n_neighbors=k, weights=weighting)
        salary_model = salary_model.fit(trainX, trainY)
        accuracy = salary_model.score(testX, testY)

        print((k, weighting, accuracy))
```

KNN Classification Results

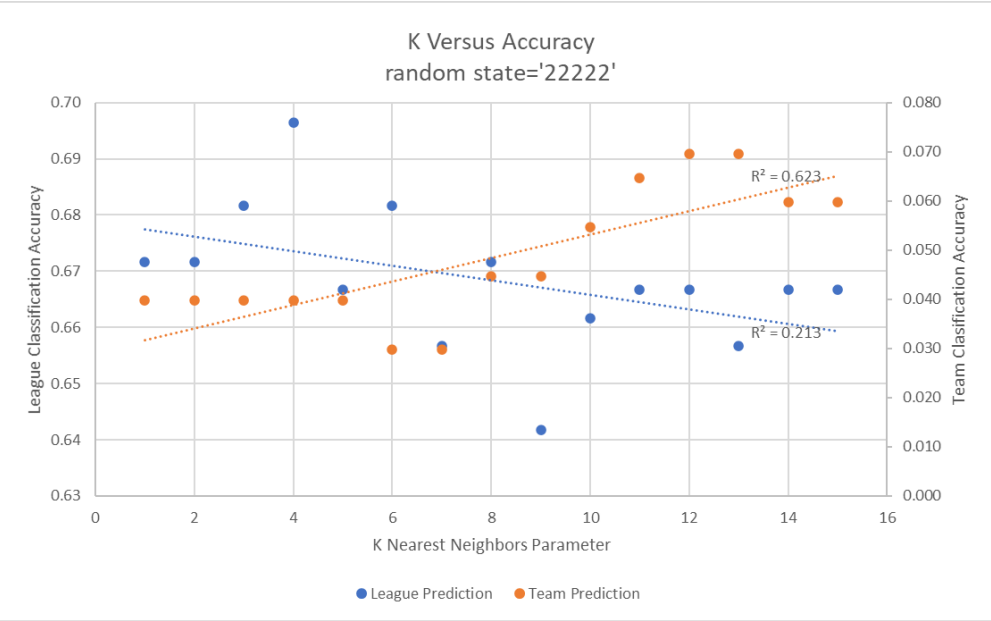
Our best classification accuracy (70%) for our KNN league prediction model had a K of 4 and used distance weighting. The average league classification accuracy of distance weighting was 66.8% compared to the average accuracy of 65.5% for uniform weighting. It makes sense that weighing neighbors by the inverse of their distance produced a more accurate model on average. In a uniform weighting scheme, the Kth furthest neighbor is weighed the same as the closest neighbor leading to lower overall accuracy.

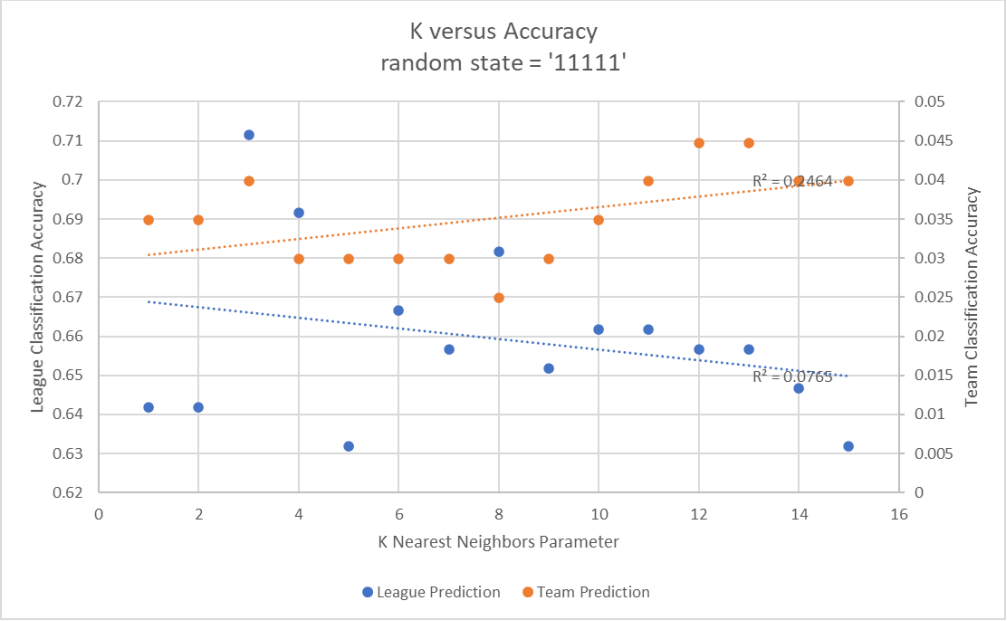
Our best classification accuracy (7%) for our team prediction model used a K of 12 and a distance weighting. It is unsurprising that we were able to predict league well but not team. There are 36 unique teams, and each team seeks the best players. It appears there wasn't a strong difference in players when separated by team giving us our low classification accuracies for team prediction. Distance weighting achieved superior accuracy over uniform weighting for the team classifier model.

League Prediction		
K	**Weight**	**Accuracy**
4	distance	0.697
3	uniform	0.682
3	distance	0.682
6	distance	0.682
1	uniform	0.672
1	distance	0.672
2	distance	0.672
8	uniform	0.672
8	distance	0.672
12	uniform	0.672
5	distance	0.667
10	uniform	0.667
11	distance	0.667
12	distance	0.667
14	distance	0.667
15	distance	0.667
10	distance	0.662
11	uniform	0.662
5	uniform	0.657
7	distance	0.657
13	distance	0.657
14	uniform	0.657
13	uniform	0.652
4	uniform	0.647
15	uniform	0.647
6	uniform	0.642
7	uniform	0.642
9	distance	0.642
2	uniform	0.632
9	uniform	0.632

Team Prediction		
K	**Weight**	**Accuracy**
12	distance	0.0697
13	distance	0.0697
11	distance	0.0647
14	distance	0.0597
15	distance	0.0597
10	distance	0.0547
2	uniform	0.0498
8	uniform	0.0498
7	uniform	0.0448
8	distance	0.0448
9	distance	0.0448
1	uniform	0.0398
1	distance	0.0398
2	distance	0.0398
3	distance	0.0398
4	uniform	0.0398
4	distance	0.0398
5	distance	0.0398
6	uniform	0.0398
9	uniform	0.0398
10	uniform	0.0398
11	uniform	0.0398
13	uniform	0.0398
5	uniform	0.0348
12	uniform	0.0348
14	uniform	0.0348
3	uniform	0.0299
6	distance	0.0299
7	distance	0.0299
15	uniform	0.0299

KNN Classification Graphs





We have a consistent outcome between our first pass using random seed ‘22222’ and our second pass using random seed ‘11111’. Best K and classification accuracies were similar. Our best K for our first random state was 4, and our second random state had a best K of 3. These are close and show that low values of K produced the best league predictions generally. Similarly, our multiclassification best value of K was 12 for the first random state and 12 for our second random state. The team prediction seemed to fair better with higher values of K, but K was not as impactful on team as on league.

There was a positive relationship between K and team prediction, and a negative relationship between K and league prediction. Our trendline confirms small neighbor amounts produced a more accurate league classification whereas higher amounts of neighbors produced a more accurate team classification. However, the relationship between team classification and number of neighbors is weak and should be investigated using other random seeds. We could also raise the ceiling for K to test if/where the classification accuracy for team begins to drop using the same seeds.

Linear Regression Code

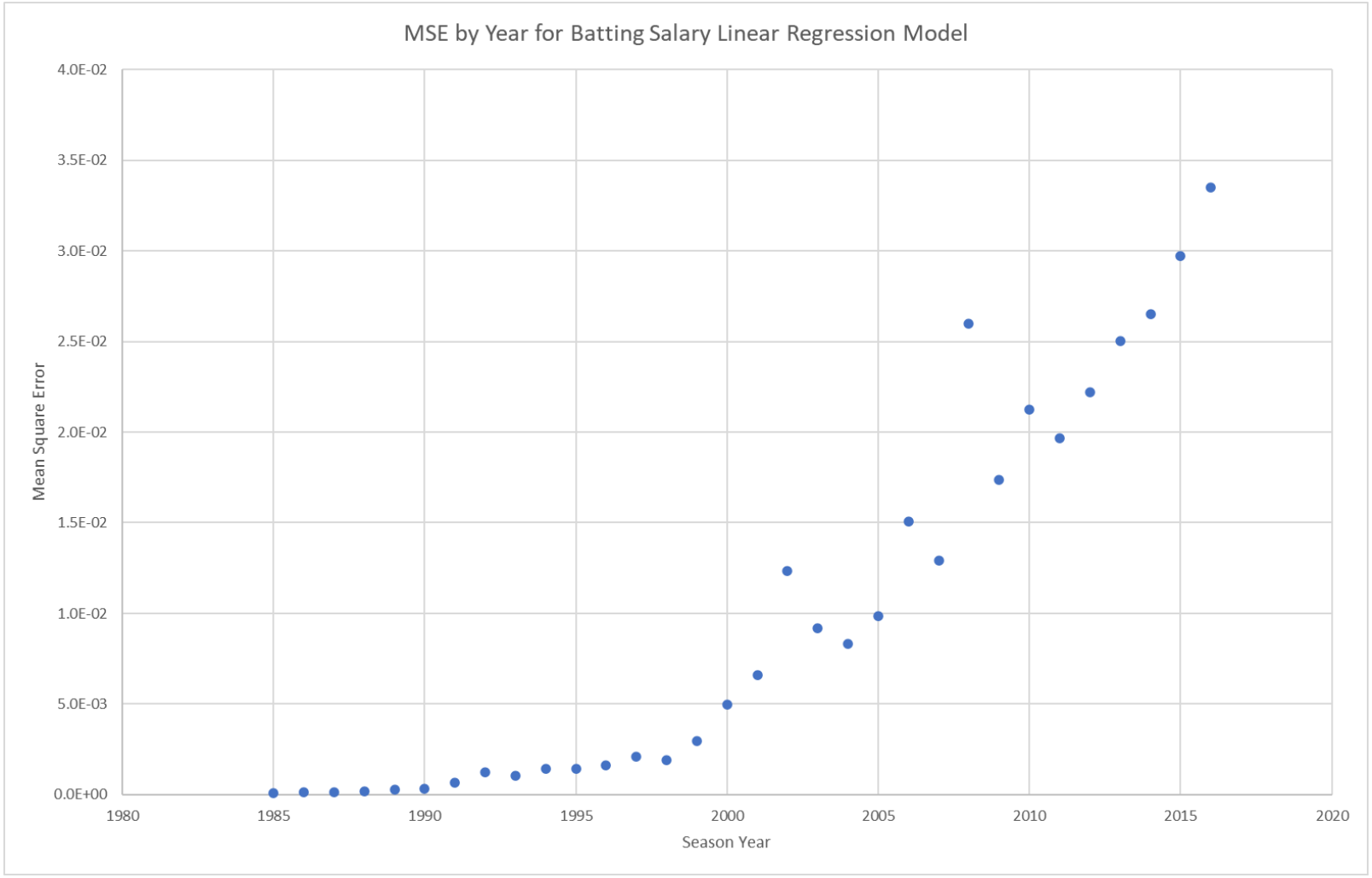
The code I developed reads the batting salaries data into a pandas data frame. I then apply pandas get dummies method to one hot encode our team ID. League is converted to binary as it only has two outcomes. We combine our one hot encoded columns with our original data frame to create the overall data set. This overall data set is split up using sklearn's model selection method with a random state of '22222' and a test size of 30% into test and training sets. After splitting we fit our model using our training features/target and record performance metrics.

We fit our model first on an overall basis and then we iterate through the data year by year to fit each model for a particular baseball season.

```
from sklearn import linear_model as linmod
import pandas as pd
import numpy as np
import sklearn as sk

filename = (r'C:\Users\Matt\Desktop\BattingSalaries_Preprocessed.xlsx')
odf= pd.read_excel(filename)

seasons= odf['yearID'].unique().tolist()
mses=[]

for season in seasons:

    results=[]
    df=odf[odf['yearID']==season]

    df=df.drop(['playerID','teamID','lgID','yearPlayer'], axis=1)
    df=(df-df.min())/(df.max()-df.min()) #normalize

    feature_names= ['G','AB','R','H','2B','3B','HR','RBI','SB','CS','BB','SO','IBB','HBP','SH','SF','GIDP']

    train_features=df[feature_names].values.tolist()
    train_target=df['Salary'].tolist()

    X = np.array(train_features)
    Y = np.array(train_target)

    (trainX, testX, trainY, testY) = sk.model_selection.train_test_split(X, Y, test_size=0.3, random_state=1)

    salary_model = linmod.LinearRegression()
    salary_model.fit(X, Y)

    R_Squared=salary_model.score(X, Y)
    print("W = ", salary_model.intercept_, salary_model.coef_, R_Squared)

    accuracy = salary_model.score(testX, testY)

    for row in testX: #predict the rain amount for each row in the test data

        prediction=salary_model.predict(row.reshape(1, -1))
        results.append(prediction[0])

    mses.append((1/len(results))*np.sum(((testY-np.array(results))**2)))
```

Linear Regression Graph

The year with the lowest MSE was 1985. As time progresses the strength of the relationship between salary and our selection of baseball statistics seems to degrade somewhat. I would imagine the underlying mechanism is that salary for professional baseball players has exploded at the top end of the skill curve and there is a resulting disconnect between skill statistics and pay which grows year over year. An average player may command only 1/100 what an elite player commands as elite players are more frequently negotiate headlining salaries. Fundamentally I believe there may be increasingly high variation in pay which I will research and discuss. Note that we did not cap salaries for this data set which is an action we can take in the future to improve model results.



Linear Regression Discussion

On our overall model the coefficient of determination was 0.264 and had an MSE of 0.0093. We modeled on 21704 events which is certainly enough events to model behavior of the overall population. The year with the strongest absolute coefficient of determination was 2008 (-.41). The year with the lowest MSE was 1985.

Surprisingly, there were many seasons with a negative relationship between salary and baseball performance statistics. What is interesting about a negative relationship between salary and performance is that it implies several highly paid elite players had underwhelming seasons.

We see great variance in our yearly coefficient of determination ranging from -0.41 to 0.34. We are likely operating on too few samples to train a robust model when we model season by season. As we are splitting our data by year, there is less data to fit our model and our models will potentially behave atypically when compared to one another or the overall model.

Perhaps clamping salaries will reduce the effect of an underwhelming player that makes dozens or hundreds of millions of dollars. The minimum MLB salary in 2021 is $570,500 in comparison to Mike Trout's $426,500,000 salary. This variance in salary increasing appears in recent seasons and likely affects model performance significantly.

Salary Prediction			
Year	**R2**	**MSE**	**Events**
Overall	**0.264**	**0.00928**	**21704**
1985	0.263	0.000095	455
1986	-0.291	0.000148	570
1987	-0.00400	0.000150	528
1988	0.133	0.000183	546
1989	-0.0174	0.000282	584
1990	-0.147	0.000345	661
1991	0.135	0.000642	541
1992	-0.0136	0.00126	582
1993	0.243	0.00106	702
1994	0.163	0.00144	655
1995	0.312	0.00142	775
1996	0.334	0.00162	737
1997	0.164	0.00208	796
1998	0.281	0.00190	871
1999	0.175	0.00298	833
2000	0.184	0.00495	757
2001	0.275	0.00658	744
2002	-0.341	0.0123	724
2003	0.204	0.00918	713
2004	0.309	0.00833	699
2005	0.169	0.00986	690
2006	-0.0469	0.0151	722
2007	0.00527	0.0129	705
2008	-0.409	0.0260	686
2009	0.109	0.0174	679
2010	-0.0112	0.0213	695
2011	0.0947	0.0197	655
2012	0.0490	0.0222	693
2013	0.0625	0.0250	667
2014	-0.0164	0.0265	694
2015	-0.00450	0.0297	677
2016	0.0942	0.0335	668