Modeling MLB Salaries

kNN Similarity and Linear Regression

John Smutny

Homework 4

ECE5984 Applications of ML

04/05/2022

Abstract

Labor costs are usually the largest expense of running a business. This is no different for each Major League Baseball (MLB) team in the United States of America with the exception that there is a cap to how much each team can spend on player salaries. This report attempts to compare the accuracy of two Machine Learning models that are trained from MLB player year, salary, and batting performance.

The first model is a kNN Similarity Classification model that is trained to predict a player's League and Team identification based on their performance and salary in 2016. The best and worst 'k' nearest neighbor term is determined for each prediction. The second model is a Linear Regression model that models a player's salary based on their affiliations and batting statistics. A model to predict salary for ALL years from 1985 to 2019 and for each individual year are detailed as well.

Part1: kNN Classification of ‘lgID’ and ‘teamID’

The goal of this model is to determine the accuracy of various k-Nearest-Neighbor classification models on their ability to classify MLB player’s League affiliation and Team affiliation in 2016. A k-value range of 1 to 15 is analyzed here with two selected random seeds: 22222 & 100. The model was trained and tested using a 70:30 training:testing ratio.

Data Preparation

The table below outlines the steps taken to ensure that the dataset used to train & test the classification models are free from errors and inconsistencies. The most influential step taken is removing all player data entries if they were not from the year 2016. As such, only 3.28% of the original dataset remains for analysis. Of the remaining entries, there were NULL values actively present in the various ‘*rat’ rate features and the salary feature. All NULL values for these features were replaced with the feature’s median values in order to outline a proper bell curve of statistics that occur naturally in a population. Please see the appendix section “*B. Initial Data Quality Report*” and “*Part 1 Supporting References*” for Data Quality Reports.

Changes to Dataset	Action Taken	# of Entries Effected	% of Entries Effected
Original dataset size:	-	45174	
2016 dataset size:	-	1483	
1. Missing data or #N/A values for non-Salary features	‘NaN’ or ‘#N/A’ values are replaced with the median value of that feature.	460	31.02%
2. Missing data or #N/A values for Salary feature	‘NaN’ or ‘#N/A’ values are replaced with the median value of that feature.	582	39.24%
3. Maximum Clamp for any Rate measurement	For all ‘*rat’ statistics; there is a domain of only [0, 1].Therefore all values greater than 1 are set to 1.	8	0.54%
4. Minimum Clamp for any Rate measurement	*see above* Therefore all values less than 0 are set to 0	0	0%
5. Positive Clamp on batting statistics.	Any non-ID, Salary, or non-*rat feature was positively clamped at 1000. The MLB record for most ‘At Bats’ (AB) in a season is 716 as of 2022. *Most instances of large values were 9999	0	0%

Table 1a: Outline of actions taken to change the used dataset and number of entries effected.

Results

Table 1b and Figures 1c & 1d show the outcome of the kNN Classification models for random seeds of 22222 and 100. Several observations can be made from relating the classification accuracy to the real world MLB teams that the model is analyzing.

As seen in Table 1b, the kNN Classification models were significantly more accurate at predicting a player's League affiliation (~59%) than a player's Team affiliation (~3.5%). Figures 1c & 1d show that in addition to being more accurate overall, the League Id models were the most accurate when the number of neighbors considered (k) increased. Compared to the Team Id Classification models, the model's accuracy consistently decreased as the number of neighbors considered increased.

	Random Seed			
	22222		**100**	
	k	**Accuracy (%)**	**k**	**Accuracy (%)**
lgID - Best k value	11	58.65	11	59.78
lgID - Worst k value	2	31.24	2	29.21
teamID - Best k value	1	3.37	1	3.82
teamID - Worst k value	2, 4 - 15	0.0	4 - 15	0

Table 1b: Summary of which Classification models were the most/least accurate.

By looking closer at Figure 1c & 1d; the accuracy of the League Id classification model did not follow a consistent positive relationship with 'k', the same way that the Team Id classification model had a negative relationship. The accuracy of the League Id classification models were significantly dependent on if the model's 'k' value was odd or even (where the odd 'k' values were significantly more accurate than the even 'k' value models).

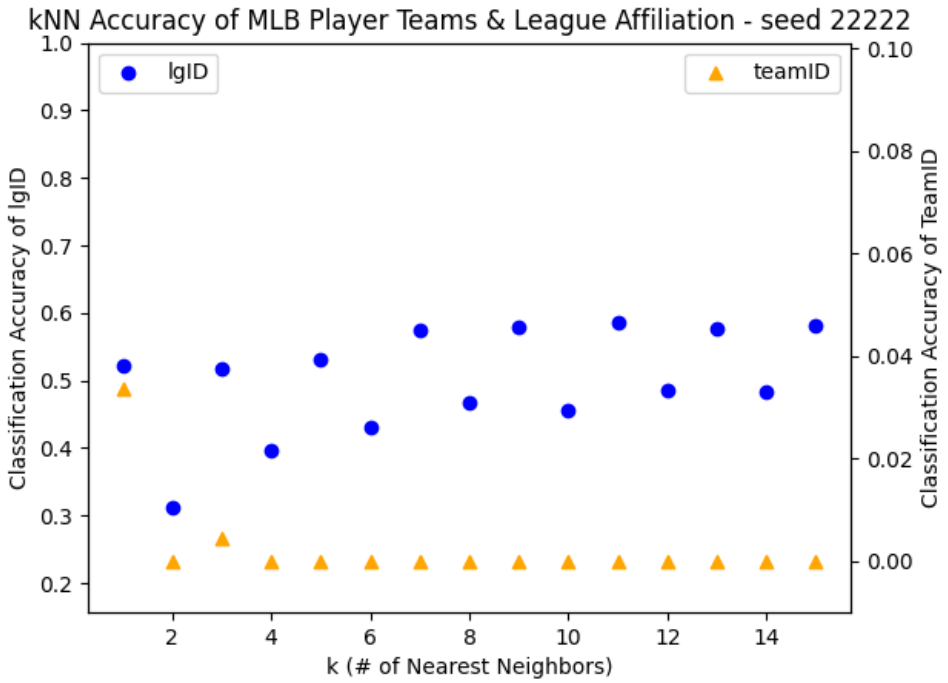

Figure 1d: Plot of classification model accuracies for varying ‘k’ values. Random seed = 22222

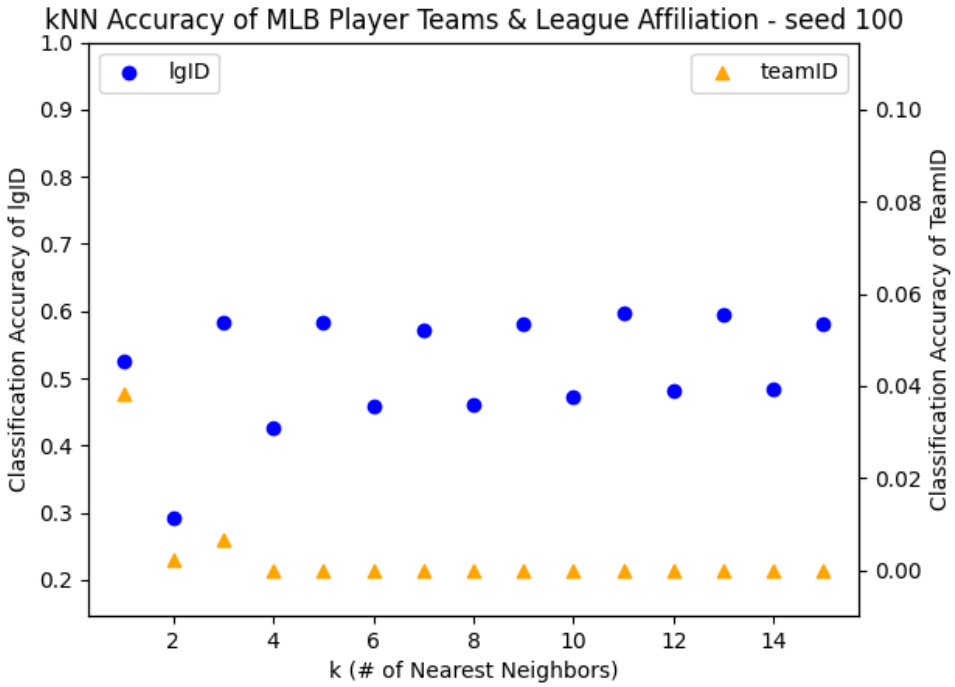

Figure 1d: Plot of classification model accuracies for varying ‘k’ values. Random seed = 100

Discussion

Prompts

1) *Choose the best value of k for each problem (binary and multiclass classification). Discuss your choice. Are they the same? Why or why not?*

Overall; the best number of neighbors to consider for the binary classification of a 2016 MLB player's League affiliation is 11, while the best number to multiclass classify a player's team is 1. See Table 1b above.

These 'k' values are not the same because of the fundamental item each model is attempting to classify. Classifying a player's League Id is a binary classification and will lead to a greater accuracy compared to attempting to classify a player's team (out of 30 teams). MLB players are all professionals and therefore it is expected that each league will have a similar make up of player skill. However, each league (National League and American League) host different rules and different stadiums that could help predict a player's league affiliation if given MORE neighbors to compare against. The opposite is true of classifying a player's team. There are 30 separate teams that are expected to all share the same bell curve of player skill. Therefore, with such a variety of categorical values to consider, it is unlikely that considering more neighbors will lead to increased accuracy. All of the MLB talent is not concentrated in one team (teams do differ, but team record was not considered) and thus makes it ideal to consider the least amount of noise as possible.

2) *Repeat the process using a different random seed value. Are the best choices for k the same as before? Why or why not?*

For both random seed values (22222 & 100), the optimal 'k' value to classify a MLB player's League and Team affiliation were the same. A 'k' value of 11 is optimal for classifying a player's league and a 'k' value of 1 is optimal for classifying a player's team.

It is suspected that the random seed did not change any optimal 'k' values for either model, because changing the random numbers used does not change the behavior of the sport of baseball or the structure of the MLB. The data is restricted by the MLB structure and thus the best 'k' value for classifications can remain consistent.

Part2: Linear Regression to Predict Player Salary

The goal of this model is to determine the accuracy of two sets of linear regression models on their ability to predict a MLB player's salary. The first model considers the range of years 1985 through 2019 in predicting salary, while the second collection of models will be predicting each year's salary in isolation (a model for 1985, a model for 1986, etc). Both sets of models will be predicting salary based on the 22222 random seed.The models were all trained and tested using a 70:30 training:testing ratio.

Data Preparation

The table below outlines the steps taken to ensure that the dataset used to train & test the classification models are free from errors and inconsistencies. Both linear regression modeling situations (predict from ALL years, vs predict by year) had the same data preparation steps applied to their relevant data.

One thing revealed by the overall data quality report in tables B-1 through B-6 in the appendix is that the full dataset from 1985 to 2019 shows a lot of the same data issues that is shown in '*Part 1: Data Preparation*' section. The same approach for each action is taken here as well, see table 2a. The biggest difference for this dataset for all recorded years is the 'teamId' cardinality and the Salary values for the years 2017, 2018, and 2019.

There are 35 teams listed in the 'teamID' feature. This is inconsistent with the fact that there are only 30 active teams in the MLB. Regardless of the inconsistency, Tables D#-# show the one-hot encoding boolean of what team each player is on. In each of these Data Quality Report sections, specifically the 'n_zero' row, there is a statistically significant number of players (over 300 entries) for each of the 35 teams. A more likely reason for the unexpected cardinality is that the team's abbreviations changed between years. Since the model goal is to model player salary, no action is taken for the 'teamId' feature.

Another observation is that all Salary values in 2017, 2018, and 2019 were null values. As such, since Salary is the target feature in this linear regression, the years 2017-2019 were excluded from this analysis.

Please see the appendix section "*Initial Data Quality Report*" and "*Part 2 Supporting References*" for other supporting tables..

Changes to Dataset	Action Taken	# of Entries Effected	% of Entries Effected
Original size:	-	45174	
1. Use One-Hot Encoding for the ‘lgID’ and ‘teamID’ categorical features.	Created 2 new binary {AL, NL} feature columns and new feature columns for each teamID cardinality	45174	100%
2. Use only numeric features (other than ‘lgID’ and ‘teamID’).	Removed the ‘playerId’ and ‘yearPlayer’ features.	45174	100%
3. Drop ALL data entries after 2016.	Since ALL Salary fields for 2017-2019 are NULL. All data entries were dropped	4597	10.18%
4. Missing data or #N/A values for non-Salary features	‘NaN’ or ‘#N/A’ values are replaced with the median value of that feature.	12230	27.07%
5. Missing data or #N/A values for Salary feature	‘NaN’ or ‘#N/A’ values are replaced with the median value for that SPECIFIC year..	12458	27.58%
6. Maximum Clamp for any Rate measurement	For all ‘*rat’ statistics; there is a domain of only [0, 1].Therefore all values greater than 1 are set to 1.	90	0.199%
6. Minimum Clamp for any Rate measurement	*see above* Therefore all values less than 0 are set to 0	0	0%
7. Positive Clamp on batting statistics.	Any non-ID, Salary, or non-*rat feature was positively clamped at 1000. The MLB record for most ‘At Bats’ (AB) in a season is 716 as of 2022. *Most instances of large values were 9999	47	0.108%
8. Data entries with Salary = 0	Any data entry with a Salary of 0 is dropped. These professionals are expected to be paid.	37	0.00819%
9. Modifications to address ‘teamID’ being not equal to 30.	No action taken. See reasoning before Table 2a.	0	0%

Table 2a: Outline of actions taken to change the used dataset and number of entries effected.

Results: Predicting Player Salary from ALL Years

When considering all years provided in the batting statistics dataset; the linear regression model's ability to predict a player's salary resulted in a r^2 score of 0.181 and a Mean Square Error score of $6448.7e^9$.

Years	r^2	Mean Square Error
1985-2019	0.25528	$6723.075e^9$

Table 2b: Overall accuracy of the linear regression model considering years 1985-2019

Results: Predicting Player Salary by Year

When a linear regression model is trained for a specific year for the years 1985-2019; the models produce varying levels of accuracy () for varying amounts of dataset values to train and test with. The most accurate (lowest MSE) model years were 2017, 2018, and 2019 with a MSE of 0.0. Note: 1987 was the most accurate non-zero MSE model year with $92.53e^9$. Meanwhile the least accurate (highest MSE) model year was 2016 with a MSE of $24199e^9$.

The number of entries considered for each model year progressively increased by year. 1985 was trained with 998 data points and 2019 was trained with the most data points, 1568. Please see appendix table D-1 for the full list of accuracy results and dataset size for each year.

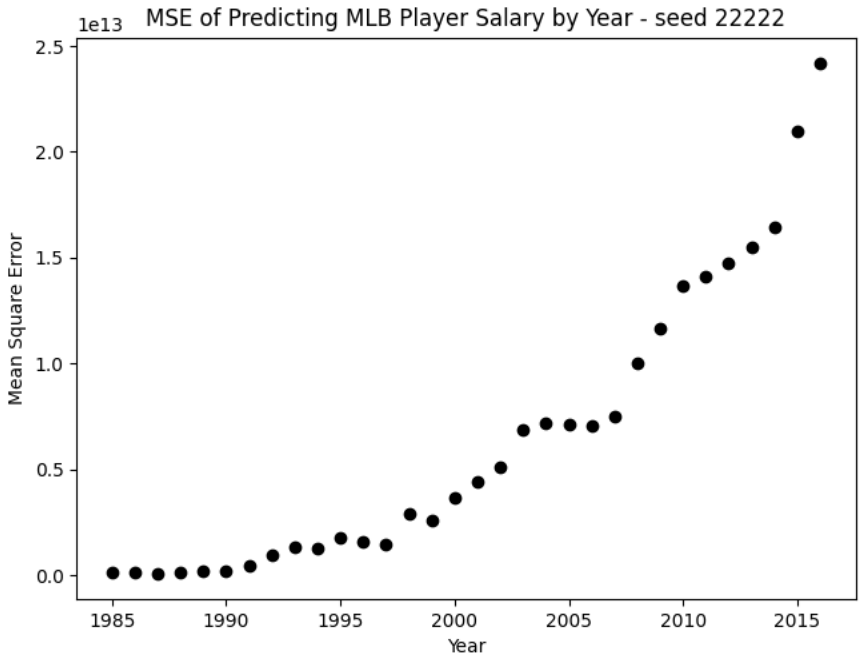

Figure 2c: Scatter plot of model accuracy by year the model was trained with.

Discussion

The results proved by Figure 2c above and table D-1 in the appendix are interesting. The results show that as time progressed the accuracy to predict a player's salary decreased until 2017 in which it was 100% accurate. It is the reporter's expectation that as time continued, accuracy would increase because there would be more data to predict from.

The alternative explanation to decreased accuracy over time is the real-world negotiations that MLB owners and MLB players have every 5 years through collective bargaining agreements (CBA). In each CBA, it is common for the salary cap, available for teams to pay for players, increases. Thus the prediction of salary from CBA to CBA will be different. This trend can be seen in small plateaus in Figure 2c, specifically in the years of 2004-06 and 2011-12. Both of these periods were in the last years of their respective CBAs [2].

The perfect accuracy results for the years 2017-2019 is concerning and would imply that every salary contract was signed predictably. Humans are inherently noisy so at least some noise is expected. One possible explanation is that owners could have begun to use their own Machine Learning models to write contracts, but that theory is flawed and is only speculative.

Further research into each individual year's data would be the next step to further refining this model's findings. Performing individualized data preparation for each individual year would most likely improve confidence in these results. In addition, to account for inflation and salary cap changes for each CBA deal, it could be more accurate to modify the salary data to be normalized in relation to the current CBA deal's maximum and minimum league wide salary (each team's salary cap times the number of teams).

Appendix

A. References

[1] BattingSalaries.xlsx. Dr Creed Jones, Virginia Tech. Data set used in all model training. 3/27/2022

[2] Nightengale, Bob. Major League Baseball, union agree on new collective bargaining agreement. USA Today. https://www.usatoday.com/story/sports/mlb/2016/11/30/mlb-collective-bargaining-agreement-lockout-union-luxury-tax/94701920/

B. Initial Data Quality Report

stat	playerID	yearID	stint	teamID	lgID	G	AB
count	45174	45174	45174	45174	45174	45174	45174
cardinality	7984	35	5	35	2	164	696
mean	*	2003.203	1.084163	*	*	54.89011	128.8025
median	*	2004	1	*	*	33	21
n_at_median	*	1346	41549	*	*	762	169
mode	*	2019	1	*	*	5	0
n_at_mode	*	1568	41549	*	*	997	12230
stddev	*	9.953198	0.292242	*	*	666.6167	833.6614
min	*	1985	1	*	*	1	0
max	*	2019	5	*	*	99999	99999
n_zero	27	N/A	N/A	1641	22861	N/A	12230
n_missing	0	0	0	0	0	0	0

Table B-1: Initial Data Quality Report describing the BattingSalaries dataset for 1985-2019

stat	R	H	2B	3B	HR	RBI	SB
count	45174	45174	45174	45174	45174	45174	45174
cardinality	147	232	61	25	66	158	82
mean	22.97355	38.47696	8.808297	5.331739	10.26168	22.12804	8.920906
median	1	3	0	0	0	1	0
n_at_median	2931	1076	23191	33343	27345	2381	31115
mode	0	0	0	0	0	0	0
n_at_mode	19793	17771	23191	33343	27345	21052	31115
stddev	815.2078	816.1902	475.2445	667.0243	814.9009	815.1981	814.9027
min	0	0	0	0	0	0	0
max	99999	99999	99999	99999	99999	99999	99999
n_zero	19793	17771	23191	33343	27345	21052	31115
n_missing	0	0	0	0	0	0	0

Table B-2: Initial Data Quality Report describing the BattingSalaries dataset for 1985-2019

stat	CS	BB	SO	IBB	HBP	SH	SF
count	45174	45174	45174	45174	45174	45174	45174
cardinality	30	149	210	44	34	30	19
mean	7.613782	18.37971	30.62549	7.573981	7.756121	7.790278	7.641365
median	0	1	6	0	0	0	0
n_at_median	32230	3144	677	33830	30721	29607	30302
mode	0	0	0	0	0	0	0
n_at_mode	32230	21084	14342	33830	30721	29607	30302
stddev	814.8917	815.0553	815.4564	814.8931	814.8911	814.8905	814.8905
min	0	0	0	0	0	0	0
max	99999	99999	99999	99999	99999	99999	99999
n_zero	32230	21084	14342	33830	30721	29607	30302
n_missing	0	0	0	0	0	0	0

Table B-3: Initial Data Quality Report describing the BattingSalaries dataset for 1985-2019

stat	GIDP	Rrat	Hrat	2Brat	3Brat	HRrat	RBIrat
count	45174	45174	45174	45174	45174	45174	45174
cardinality	34	7211	7438	5032	2327	5369	7776
mean	2.732191	0.104367	0.202044	0.036848	0.003911	0.01821	0.091622
median	0	0.106796	0.230769	0.037037	0	0.007944	0.08971
n_at_median	25226	14	142	134	21113	N/A	2
mode	0	0	0	0	0	0	0
n_at_mode	25226	7632	5541	10961	21113	15115	8828
stddev	4.656144	0.117684	0.131607	0.047444	0.015431	0.031195	0.099829
min	0	0	0	0	0	0	0
max	35	5	1	1	1	1	4
n_zero	25226	7632	5541	10961	21113	15115	8828
n_missing	0	12230	12230	12230	12230	12230	12230

Table B-4: Initial Data Quality Report describing the BattingSalaries dataset for 1985-2019

stat	SBrat	CSrat	BBrat	SOrat	IBBrat	HBPrat	SHrat
count	45174	45174	45174	45174	45174	45174	45174
cardinality	4677	2927	7772	9606	2951	3113	3044
mean	0.013014	0.006094	0.076968	0.288368	0.004374	0.007532	0.039519
median	0	0	0.067093	0.228571	0	0	0
n_at_median	18892	20005	2	26	21600	18496	17513
mode	0	0	0	0	0	0	0
n_at_mode	18892	20005	8952	2112	21600	18496	17513
stddev	0.056136	0.027715	0.098545	0.224484	0.014238	0.028056	0.130749
min	0	0	0	0	0	0	0
max	5	2	2	1	1	1	3
n_zero	18892	20005	8952	2112	21600	18496	17513
n_missing	12230	12230	12230	12230	12230	12230	12230

Table B-5: Initial Data Quality Report describing the BattingSalaries dataset for 1985-2019

stat	SFrat	GIDPrat	yearPlayer	Salary
count	45174	45174	45174	45174
cardinality	2555	4179	41549	3352
mean	0.006208	0.020004	*	2122648
median	0	0.013846	*	600000
n_at_median	18074	N/A	*	223
mode	0	0	*	109000
n_at_mode	18074	12996	*	644
stddev	0.022546	0.04632	*	3448148
min	0	0	*	0
max	1	1	*	33000000
n_zero	18074	12996	5	1
n_missing	12230	12230	0	17055

Table B-6: Initial Data Quality Report describing the BattingSalaries dataset for 1985-2019

C. Part 1 Supporting References

Additional data quality reports after data processing steps are available by request

D. Part 2 Supporting References

Additional data quality reports after data processing steps are available by request

YearID	Dataset Size	r2	Mean Square Error
1985	998	-1.19575	1.39E+11
1986	1017	-0.06327	1.22E+11
1987	1048	0.061434	9.25E+10
1988	1035	0.073008	1.14E+11
1989	1073	0.063432	2.24E+11
1990	1115	0.185201	2.08E+11
1991	1086	0.133234	4.19E+11
1992	1066	0.144856	9.72E+11
1993	1179	0.013974	1.3E+12
1994	1030	0.211785	1.28E+12
1995	1253	0.243821	1.78E+12
1996	1253	0.322911	1.59E+12
1997	1236	0.114301	1.44E+12
1998	1322	0.244635	2.9E+12
1999	1299	0.268674	2.58E+12
2000	1384	0.16189	3.65E+12
2001	1339	0.300959	4.41E+12
2002	1319	0.203681	5.14E+12
2003	1347	0.317046	6.89E+12
2004	1346	0.24254	7.21E+12
2005	1330	0.249886	7.13E+12
2006	1377	0.327638	7.07E+12
2007	1385	0.097379	7.49E+12

2008	1385	0.258251	1E+13
2009	1388	0.119116	1.16E+13
2010	1356	0.12749	1.37E+13
2011	1389	0.162181	1.41E+13
2012	1408	0.126224	1.48E+13
2013	1409	0.15779	1.55E+13
2014	1435	-0.00656	1.64E+13
2015	1486	0.062054	2.1E+13
2016	1483	0.124794	2.42E+13
2017	0	-	-
2018	0	-	-
2019	0	-	-

Table D-1: Full table of linear regression model accuracy and dataset considered by year.

Python Code

Part1: kNN Classification of ‘lgID’ and ‘teamID’

```
'''
@package    HW4-Similarity&linearRegression
@fileName   hw4_ClassifyPlayer-TeamAndLeague
@author     John Smutny
@date       04/03/2022
@info       Create a script that will predict a 2016 player's league
           affiliation and team affiliation based on the player's batting
           statistics & salary by using a k-NearestNeighbor Classification
           model.
'''

# Python Libraries

# Third Party libraries
from sklearn import neighbors
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Local libraries
from vt_ApplicationsOfML.Libraries.DataExploration.DataQualityReport import \
   DataQualityReport

## Control flags and constants
###############################################################################
##
DEBUG = False
OUTPUT_FILES = True
TRAIN_RATIO = 0.7
#RANDOM_SEED = 22222
RANDOM_SEED = 100

# Establish range of k values to compare similarity model accuracy.
MIN_K = 1
MAX_K_exclusive = 16

INPUT_FILE = '../Info&Data/BattingSalaries.xlsx'
OUTPUT_DQR1 = '../artifacts/BattingSalaries_1_DQR1.xlsx'
OUTPUT_DQR2 = '../artifacts/BattingSalaries_1_DQR2.xlsx'
OUTPUT_DQR3 = '../artifacts/BattingSalaries_1_DQR3_Normalized.xlsx'
OUTPUT_FILE = '../artifacts/BattingSalaries_EDIT.xlsx'
OUTPUT_IMAGE = '../artifacts/kNN_Results-seed{}.png'.format(
   RANDOM_SEED)
```

```
# Validate control flags
assert DEBUG in [0, 1, True, False,
                 None], 'DEBUG flag must be a valid true-false value'
assert OUTPUT_FILES in [0, 1, True, False,
                        None], 'OUTPUT_FILES flag must be a valid binary value'
assert MIN_K in [MIN_K < MAX_K_exclusive], 'min K must be smaller than max'
assert MAX_K_exclusive in range(1, 17), 'Only valid range 1-15'

## Helper Functions
###############################################################################
##
FEATURES_TO_NORMALIZE = [
    'G',
    'AB',
    'R',
    'H',
    '2B',
    '3B',
    'HR',
    'RBI',
    'SB',
    'CS',
    'BB',
    'SO',
    'IBB',
    'HBP',
    'SH',
    'SF',
    'GIDP',
    'Salary'
]


## Data Handling Functions
###############################################################################
##

def cleanRawData(df: pd.DataFrame) -> pd.DataFrame:
    ''' Clean undesired data from the inputted data '''
    print("Begin Data Preparation...")

    # Keep track of number of values effected by each step
    orig_size = len(df)
    numNULL = 0
    numRatNULL = 0
    numAbove1 = 0
    numBelow0 = 0
    numHigh = 0
```

```
    # 2) Drop the quality_flag & source_flag column
    df = df.drop(columns=['playerID', 'yearPlayer'])


    for label in df.columns:
        col = df[label]

        # Refine the 'rate' values
        if 'rat' in label:
            # 3) Set a [0, 1] clamp on every 'Rate' feature (*rat in the name)
            if col.max() > 1:
                numAbove1 = numAbove1 + np.sum(col > 1)
                col.where(col <= 1, 1, inplace=True)
            if col.min() < 0:
                numBelow0 = numBelow0 + np.sum(col < 0)
                col.where(col >= 0, 0, inplace=True)

            # 4) Replace all null 'Rate' values with zero (*rat in the name).
            if col.isnull().sum() != 0:
                numRatNULL = numRatNULL + col.isnull().sum()
                df[label] = col.fillna(0)

        # 5) Replace non-rat NULL or #N/A values with the median of that
feature
        elif col.isnull().sum() != 0:
            numNULL = numNULL + col.isnull().sum()
            medianV = col.median()
            df[label] = col.fillna(medianV)

        # 5) Check for invalidly high values (>1000) replace with 0
        if type(col[col.first_valid_index()]) != str and \
                label != 'Salary' and label != 'yearID':
            numHigh = numHigh + np.sum(col > 1000)
            col.where(col < 1000, 0, inplace=True)

    print("Data Prep # Changed Values out of {}:\n#NULL: {}"
          "\n#AboveOne: {}\n#BelowZero: {}\n#HighValues: {}".format(
                                                orig_size, numNULL,
                                                numAbove1, numBelow0, numHigh))

    print("Data Preparation COMPLETE...")

    return df


## Main Data Processing
#############################################################################
##
```

```
# Load and analyze
df_raw = pd.read_excel(INPUT_FILE, sheet_name='Batting')

# Data Preparation
# Remove all entries that are not from the 2016 season
df_raw = df_raw.query("yearID == 2016")
report1 = DataQualityReport()
report1.quickDQR(df_raw, list(df_raw.columns))

df_stats = cleanRawData(df_raw)
report2 = DataQualityReport()
report2.quickDQR(df_stats, list(df_stats.columns))

# Normalize all numeric features.
for label in FEATURES_TO_NORMALIZE:
   df_stats[label] = df_stats[label] / df_stats[label].max()
report3 = DataQualityReport()
report3.quickDQR(df_stats, list(df_stats.columns))

if OUTPUT_FILES:
   report1.to_excel(OUTPUT_DQR1)
   report2.to_excel(OUTPUT_DQR2)
   report3.to_excel(OUTPUT_DQR3)

################################
# Create kNN Model
training = df_stats.sample(frac=TRAIN_RATIO, random_state=RANDOM_SEED)
test = df_stats.drop(training.index)

# Get Training set and One-Hot Encoding features
x_train = training.drop(columns=['teamID', 'lgID'])
y_train_lgId = pd.get_dummies(training.lgID, prefix='lg')
y_train_teamID = pd.get_dummies(training.teamID, prefix='teamID')

# Get Test set and One-Hot Encoding features
x_test = test.drop(columns=['teamID', 'lgID'])
y_test_lgID = pd.get_dummies(test.lgID, prefix='lg')
y_test_teamID = pd.get_dummies(test.teamID, prefix='teamID')

################################
# Train the 'lgID' and 'TeamId' classifiers and calculate accuracy
rangeK = range(MIN_K, MAX_K_exclusive)
lgId_accuracy = np.zeros(len(rangeK))
teamId_accuracy = np.zeros(len(rangeK))

for i in range(MAX_K_exclusive - MIN_K):
   k = i + MIN_K
   print("Training Model k = {} out of {} Starting...".format(k,
```

```
                                                                MAX_K_exclusive -
1))

   clf = neighbors.KNeighborsClassifier(n_neighbors=k, weights='uniform')
   clf.fit(x_train, y_train_lgId)
   result_lgId = clf.predict(np.asarray(x_test))
   lgId_accuracy[i] = clf.score(x_test, y_test_lgID)

   clf = neighbors.KNeighborsClassifier(n_neighbors=k, weights='uniform')
   clf.fit(x_train, y_train_teamID)
   result_teamId = clf.predict(np.asarray(x_test))
   teamId_accuracy[i] = clf.score(x_test, y_test_teamID)

################################
# Display Results
best_k_lgId = np.where(lgId_accuracy == max(lgId_accuracy))
best_k_teamId = np.where(teamId_accuracy == max(teamId_accuracy))
print("RESULTS: Best (k/accuracy):\n\tlgId = {}/{}\n\tteamId = {},"
     "{}".format(best_k_lgId[0]+1, max(lgId_accuracy),
                 best_k_teamId[0]+1, max(teamId_accuracy)))

worst_k_lgId = np.where(lgId_accuracy == min(lgId_accuracy))
worst_k_teamId = np.where(teamId_accuracy == min(teamId_accuracy))
print("RESULTS: Worst (k/accuracy):\n\tlgId = {}/{}\n\tteamId = {},"
     "{}".format(worst_k_lgId[0]+1, min(lgId_accuracy),
                 worst_k_teamId[0]+1, min(teamId_accuracy)))

# Plot Classification Accuracy chart of both features 'lgID' and 'teamID'.
fig, scat1 = plt.subplots()

scat1.scatter(rangeK, lgId_accuracy, marker='o', c='blue', label='lgID')
scat1.set_ylim([min(lgId_accuracy)*0.50, 1])
scat1.set_xlabel("k (# of Nearest Neighbors)")
scat1.set_ylabel("Classification Accuracy of lgID")
scat1.legend(loc='upper left')

scat2 = scat1.twinx()
scat2.scatter(rangeK, teamId_accuracy, marker='^', c='orange', label='teamID')
scat2.set_ylim([-0.01, max(teamId_accuracy)*3])
scat2.set_ylabel("Classification Accuracy of TeamID")
plt.title("kNN Accuracy of MLB Player Teams & League Affiliation - seed "
          "{}".format(RANDOM_SEED))
scat2.legend(loc='upper right')
plt.show()

if OUTPUT_FILES:
   fig.savefig(OUTPUT_IMAGE, format='png', dpi=100)
```

Part2: Linear Regression to predict Player Salaries

```
'''
@package    HW4-Similarity&linearRegression
@fileName   hw4_LinearRegression-PredictYearlySalary
@author     John Smutny
@date       04/03/2022
@info       Create a script that will predict the salary of a player for any
           given year included in the dataset. Use a multivariate linear
           regression model to predict the salary for each year.

           Accuracy for each year's prediction is measured by r2 and mean
           square error values.
'''

# Third Party libraries
import numpy as np
import pandas as pd
from sklearn import linear_model as linmod
from sklearn import metrics
import matplotlib.pyplot as plt

# Local libraries
from vt_ApplicationsOfML.Libraries.DataExploration.DataQualityReport import \
   DataQualityReport

## Control flags and constants
##############################################################################
##
DEBUG = False
OUTPUT_FILES = True
TRAIN_RATIO = 0.7
RANDOM_SEED = 22222
#RANDOM_SEED = 100

INPUT_FILE = '../Info&Data/BattingSalaries.xlsx'
OUTPUT_DQR1 = '../artifacts/BattingSalaries_2_DQR1.xlsx'
OUTPUT_DQR2 = '../artifacts/BattingSalaries_2_DQR2.xlsx'
OUTPUT_DQR3 = '../artifacts/BattingSalaries_2_DQR3_OneHotEncoding.xlsx'
OUTPUT_R2 = '../artifacts/results_r2.xlsx'
OUTPUT_MSE = '../artifacts/results_mse.xlsx'
OUTPUT_IMAGE = '../artifacts/linearRegressionResults-seed{}.png'.format(
   RANDOM_SEED)

## Helper Functions
##############################################################################
##
```

```
TARGET_FEATURE = 'Salary'

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


## Data Handling Functions
#############################################################################
##

def cleanRawData(df: pd.DataFrame) -> pd.DataFrame:
    ''' Clean undesired data from the inputted data '''
    print("Begin Data Preparation...")

    # Keep track of number of values effected by each step
    orig_length = len(df)
    orig_size = len(df) * len(df.columns)
    numDropped = 0
    numRatNULL = 0
    numSalaryPost2016 = 0
    numSalaryNULL = 0
    numAbove1 = 0
    numBelow0 = 0
    numHigh = 0

    # 1) Drop the quality_flag & source_flag column
    df = df.drop(columns=['playerID', 'yearPlayer'])
    numDropped = orig_size - len(df) * len(df.columns)

    # 2) Drop any player with salary of zero.
    numDropped = numDropped + len(df[df['Salary'] == 0].sum())
```

```
df = df.query("Salary != 0")

# 2) Drop all data entries 2017 & later
numSalaryPost2016 = numSalaryPost2016 + len(df.query("yearID >= 2017"))

# 3) Replace any NULL salaries with that year's median
df.query("yearID < 2017", inplace=True) #TODO - TEST
numSalaryNULL = numSalaryNULL + df['Salary'].isnull().sum()

for yr in df['yearID'].unique():
    medianV = df[df['yearID'] == yr]['Salary'].median()
    yrSalary = df[df['yearID'] == yr]['Salary']
    df.loc[df['yearID'] == yr, 'Salary'] = yrSalary.fillna(medianV)

# Refine values based on the batting statistic
for label in df.columns:
    col = df[label]

    # Refine the 'rate' values
    if 'rat' in label:
        # 4) Set a [0, 1] clamp on every 'Rate' feature (*rat in the name).
        if col.max() > 1:
            numAbove1 = numAbove1 + np.sum(col > 1)
            col.where(col <= 1, 1, inplace=True)
        if col.min() < 0:
            numBelow0 = numBelow0 + np.sum(col < 0)
            col.where(col >= 0, 0, inplace=True)

        # 5) Replace all null 'Rate' values with zero (*rat in the name).
        if col.isnull().sum() != 0:
            numRatNULL = numRatNULL + col.isnull().sum()
            df[label] = col.fillna(0)

    # 6) Check for invalidly high values (>1000) replace with 0
    if type(col[col.first_valid_index()]) != str and \
            label != 'Salary' and label != 'yearID':
        numHigh = numHigh + np.sum(col > 1000)
        col.where(col < 1000, 0, inplace=True)

print("Data Prep # Changed Values out of {}:\n#Dropped: {}\n#RatNULL: {}"
      "\n#SalaryNULL: {}\n#SalaryPost2016: {}\n"
      "#AboveOne: {}\n#BelowZero: {}\n#HighValues: {}".format(
                                    orig_size, numDropped, numRatNULL,
                                    numSalaryNULL, numSalaryPost2016,
                                    numAbove1, numBelow0, numHigh))

print("Data Preparation COMPLETE...")

return df
```

```
## Main Data Processing
################################################################################
##

# Load and analyze
df_raw = pd.read_excel(INPUT_FILE, sheet_name='Batting')
report1 = DataQualityReport()
report1.quickDQR(df_raw, list(df_raw.columns))

# Data Preparation
df_stats = cleanRawData(df_raw)
report2 = DataQualityReport()
report2.quickDQR(df_stats, list(df_stats.columns))

# Apply One-Hot Encoding Features [lgId, TeamId]
# Convert categorical 'lgId' and 'teamId' via One-Hot Encoding
col_lgId = pd.get_dummies(df_stats.lgID, prefix='lg')
col_teamID = pd.get_dummies(df_stats.teamID, prefix='teamID')

df_stats.drop(columns=['lgID', 'teamID'], inplace=True)
df_stats = df_stats.join(col_lgId)
df_stats = df_stats.join(col_teamID)

report3 = DataQualityReport()
report3.quickDQR(df_stats, list(df_stats.columns))

if OUTPUT_FILES:
   report1.to_excel(OUTPUT_DQR1)
   report2.to_excel(OUTPUT_DQR2)
   report3.to_excel(OUTPUT_DQR3)

################################
# Create Linear Regression model for ALL years
training = df_stats.sample(frac=TRAIN_RATIO, random_state=RANDOM_SEED)
test = df_stats.drop(training.index)

train_x = training.drop(columns=[TARGET_FEATURE]).to_numpy()
train_y = training[TARGET_FEATURE].to_numpy()
test_x = test.drop(columns=[TARGET_FEATURE]).to_numpy()
test_y = test[TARGET_FEATURE].to_numpy()

mlr = linmod.LinearRegression()
mlr.fit(train_x, train_y)

# Determine Accuracy measures (r2 & MSE)
r2 = mlr.score(test_x, test_y)
mse = metrics.mean_squared_error(test_y, mlr.predict(test_x))
```

```
print("LinearRegression r2 & MSE: {} & {}".format(r2, mse))

################################
# Create Linear Regression model for each year
r2_by_year = {}
mse_by_year = {}

for yr in df_stats['yearID'].unique():
   dataYr = df_stats[df_stats['yearID'] == yr]

   training = dataYr.sample(frac=TRAIN_RATIO, random_state=RANDOM_SEED)
   test = dataYr.drop(training.index)

   train_x = training.drop(columns=[TARGET_FEATURE]).to_numpy()
   train_y = training[TARGET_FEATURE].to_numpy()
   test_x = test.drop(columns=[TARGET_FEATURE]).to_numpy()
   test_y = test[TARGET_FEATURE].to_numpy()

   mlr = linmod.LinearRegression()
   mlr.fit(train_x, train_y)

   # Determine Accuracy measures (r2 & MSE)
   r2_by_year[yr] = mlr.score(test_x, test_y)
   mse_by_year[yr] = metrics.mean_squared_error(test_y, mlr.predict(test_x))
   print("Accuracy: {} - {}/{} from {}".format(yr, r2_by_year[yr],
                                       mse_by_year[yr], len(dataYr)))

# Publish 'accuracy by year' results
if OUTPUT_FILES:
   df_r2 = pd.DataFrame(data=r2_by_year, index=[0])
   df_r2.to_excel(OUTPUT_R2)
   df_mse = pd.DataFrame(data=mse_by_year, index=[0])
   df_mse.to_excel(OUTPUT_MSE)

################################
# Plot MSE over year to determine the most predictable salary year.
plt.plot(mse_by_year.keys(), list(mse_by_year.values()), 'o',
        color='black')
plt.title("MSE of Predicting MLB Player Salary by Year - seed {}".format(
                                                                RANDOM_SEED))
plt.xlabel("Year")
plt.ylabel("Mean Square Error")
plt.show()
```