MVB_proj_CS105

March 19, 2020

0.1 Bhagpreet Brar

0.2 SID: 862013514

My first question/hypothesis I have about my data is what types of features relate to the number of playoff wins on the current season based on the stats of the regular season and possibly overall totals on coaches previous playoff stats and records. The next question/hypothesis is determining whether or not I can build an efficient model to determine a coaches playoff performance, possibly win percent or number of games they'll win in that current playoff season. This hypothesis will be determined on a specific coach that has a variety of years of stats to use.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv("Integrated_NBA_Coach_Stats.csv")
df = df.drop(columns = ['Unnamed: 0', 'Unnamed: 0.1', 'Seasons'])
df = df.fillna(0)
df = pd.get_dummies(df)
df
```

[1]:

	numSznsWTeam	numTotSzns	curr_games	curr_wins	curr_losses	\
0	13	17	80	58	22	
1	1	1	38	12	26	
2	3	5	80	48	32	
3	7	7	82	36	46	
4	5	5	82	42	40	
…	…	…	…	…	…	
1050	2	15	82	35	47	
1051	1	1	82	31	51	
1052	5	5	82	42	40	
1053	13	20	69	32	37	
1054	1	14	13	7	6	

	franch_games	franch_wins	franch_losses	tot_games	tot_wins	…	\
0	952	620	332	1177	763	…	
1	38	12	26	38	12	…	
2	239	127	112	342	183	…	
3	536	291	245	536	291	…	

```
4              384          201            183         384        201  …
…              …            …              …           …          …  …
1050           164           76             88        1222        588  …
1051            82           31             51          82         31  …
1052           390          175            215         390        175  …
1053          1027          522            505        1525        757  …
1054            13            7              6         987        459  …

      Team_OKC  Team_ORL  Team_PHI  Team_PHO  Team_POR  Team_SAC  Team_SAS  \
0            0         0         0         0         0         0         0
1            0         0         0         0         0         0         0
2            0         0         1         0         0         0         0
3            0         0         0         0         0         0         0
4            0         0         0         0         0         0         0
…        …       …         …         …         …         …         …
1050         0         0         0         0         0         0         1
1051         1         0         0         0         0         0         0
1052         0         0         0         0         0         0         0
1053         0         0         0         0         0         0         0
1054         0         0         0         0         0         0         0

      Team_TOR  Team_UTA  Team_WAS
0            0         0         0
1            0         0         1
2            0         0         0
3            0         0         0
4            0         0         1
…        …       …         …
1050         0         0         0
1051         0         0         0
1052         0         1         0
1053         0         0         1
1054         0         0         1

[1055 rows x 128 columns]
```

[2]:
```
from sklearn.linear_model import LinearRegression

df['win_percent_squared'] = df['win_percent'] * df['win_percent']
X_train = df[df['Season'] <= 2009]
X_test = df[df['Season'] >= 2010]

model = LinearRegression()

model.fit(
```

```
    X = X_train[['curr_wins', 'tot_wins', 'franch_wins', 'win_percent',␣
 ↪'win_percent_squared', 'tot_pl_games', 'tot_pl_wins', 'Season',␣
 ↪'franch_pl_wins']],
    y = X_train['curr_pl_wins']
)

y_pred = model.predict(
    X = X_test[['curr_wins', 'tot_wins', 'franch_wins','win_percent',␣
 ↪'win_percent_squared', 'tot_pl_games', 'tot_pl_wins', 'Season',␣
 ↪'franch_pl_wins']]
)

model.coef_
```

[2]:
```
array([ 1.33455122e-01, -7.10682164e-04, -1.59681855e-02, -1.34194523e+01,
        1.61972472e+01,  2.16212479e-02, -3.26841834e-02, -5.04542690e-03,
        1.60087729e-01])
```

In this first model, I wanted to determine what are the best coefficients to determine the number playoff season wins for the specific coach and teams. Currently we arent using the coaches and teams to influence or regression line but will be included in the next step along with validation. Some of the coefficients are data on the total wins the coach currently has, to different things like the coaches total playoff games coached and number of total games and wins and such. From phase 2, using EDA we found out that the win_percent column is negatively skewed, so in the testing of coefficients I squared the column to normalize the values and created a new column for it, and it gave us the best related coefficient to the number of current playoff season wins. The win_percent column is the win percent of the current regular season of the team.

Here are the top 3 coefficients that are closely related to our target playoff season win percentage: 1. win_percent_squared | coef: 16.197 (normalized regular season win percent) 2. franch_pl_wins | coef: 0.160 (total playoff games won by the specific team and coach) 3. curr_wins | coef: 0.133 (current regular season wins of the coach's season)

Here are the top 3 coefficients that are unrelated to our target playoff season win percentage: 1. win_percent | coef: -13.419 2. tot_pl_wins | coef: -0.033 (Coach's overall playoff games won so far) 3. franch_wins | coef: -0.016

[3]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
```

```
X_train = df[df['Season'] <= 2009].drop(columns = ['win_percent',␣
 ↪'tot_pl_wins', 'numSznsWTeam', 'numTotSzns', 'curr_games', 'curr_losses',␣
 ↪'franch_games', 'franch_losses', 'tot_games', 'tot_losses', 'curr_pl_games',␣
 ↪'curr_pl_wins', 'curr_pl_losses', 'franch_pl_games', 'franch_pl_losses',␣
 ↪'tot_pl_games', 'tot_pl_losses', 'pl_win_percent'])
X_test = df[df['Season'] >= 2010].drop(columns = ['win_percent', 'tot_pl_wins',␣
 ↪'numSznsWTeam', 'numTotSzns', 'curr_games', 'curr_losses', 'franch_games',␣
 ↪'franch_losses', 'tot_games', 'tot_losses', 'curr_pl_games', 'curr_pl_wins',␣
 ↪'curr_pl_losses', 'franch_pl_games', 'franch_pl_losses', 'tot_pl_games',␣
 ↪'tot_pl_losses', 'pl_win_percent'])
y_train = df[df['Season'] <= 2009]['curr_pl_wins']
y_test = df[df['Season'] >= 2010]['curr_pl_wins']
full_set_X = df.drop(columns = ['win_percent', 'tot_pl_wins', 'numSznsWTeam',␣
 ↪'numTotSzns', 'curr_games', 'curr_losses', 'franch_games', 'franch_losses',␣
 ↪'tot_games', 'tot_losses', 'curr_pl_games', 'curr_pl_wins',␣
 ↪'curr_pl_losses', 'franch_pl_games', 'franch_pl_losses', 'tot_pl_games',␣
 ↪'tot_pl_losses', 'pl_win_percent'])
full_set_y = df['curr_pl_wins']

pipeline = make_pipeline(
    StandardScaler(),
    SGDRegressor()
)

pipeline.fit(X = X_train, y = y_train)

RMSE = cross_val_score(pipeline,
                       X = full_set_X,
                       y = full_set_y,
                       cv = 5)  # cv refers to number of cross-validation cuts
MSE = RMSE*RMSE

print('Mean of MSE\'s: ' + str(np.mean(MSE)))
print('Median of MSE\'s: ' + str(np.median(MSE)) + '\n')

y_pred = pipeline.predict(X=X_test)

print('Single sample prediction value: ' + str(y_pred[y_pred.size - 2]))
print('Single sample actual value: ' + str(y_test[735])) #index here is weird␣
 ↪because we split the y_test from entire dataset
print('\nVALUES CHANGE ON EACH RUN OF THIS CODE')
```

```
Mean of MSE's: 0.21850579461113256
Median of MSE's: 0.2119642443573018

Single sample prediction value: 2.41118491574027
Single sample actual value: 3.0
```

VALUES CHANGE ON EACH RUN OF THIS CODE

For the code above, we used the best coefficients along with the one hot encoding of the coach and teams categories. We build a pipeline with standard scaler for data and using a SGD regressor to fit the data and perform the gradient descent to minimize error. On the entire dataset, a 5 fold cross validation is ran, and the 5 RMSE's are calculated for the 5 fold cross validation. After squaring the RMSE to get the MSE, the MSE mean and median is around 0.21 . That is a small and good margin of error considering that our prediction values are within the range of 0 to 16. The single sample prediction we observe in the ran code seems very close to the actual value, having not too bad of an error margin.

[4]:
```
PJ_data = df[df['Coach_Phil Jackson'] == 1].drop(columns = ['win_percent',␣
 ↪'tot_pl_wins', 'numSznsWTeam', 'numTotSzns', 'curr_games', 'curr_losses',␣
 ↪'franch_games', 'franch_losses', 'tot_games', 'tot_losses', 'curr_pl_games',␣
 ↪'curr_pl_losses', 'franch_pl_games', 'franch_pl_losses', 'tot_pl_losses',␣
 ↪'pl_win_percent'])
PJ_data = PJ_data.loc[:, ~PJ_data.columns.str.startswith('Coach')]# Seasons␣
 ↪Coached: 20 (1989-2010)

GP_data = df[df['Coach_Gregg Popovich'] == 1].drop(columns = ['win_percent',␣
 ↪'tot_pl_wins', 'numSznsWTeam', 'numTotSzns', 'curr_games', 'curr_losses',␣
 ↪'franch_games', 'franch_losses', 'tot_games', 'tot_losses', 'curr_pl_games',␣
 ↪'curr_pl_losses', 'franch_pl_games', 'franch_pl_losses', 'tot_pl_losses',␣
 ↪'pl_win_percent'])
GP_data = GP_data.loc[:, ~GP_data.columns.str.startswith('Coach')]# Seasons␣
 ↪Coached: 23 (1996-2018)

PR_data = df[df['Coach_Pat Riley'] == 1].drop(columns = ['win_percent',␣
 ↪'tot_pl_wins', 'numSznsWTeam', 'numTotSzns', 'curr_games', 'curr_losses',␣
 ↪'franch_games', 'franch_losses', 'tot_games', 'tot_losses', 'curr_pl_games',␣
 ↪'curr_pl_losses', 'franch_pl_games', 'franch_pl_losses', 'tot_pl_losses',␣
 ↪'pl_win_percent'])
PR_data = PR_data.loc[:, ~PR_data.columns.str.startswith('Coach')]# Seasons␣
 ↪Coached: 24 (1981-2007)

#PJ_train_X = PJ_data[PJ_data['Season'] <= 1995].drop(columns =␣
 ↪['curr_pl_wins'])
#y_PJ_train = PJ_data[PJ_data['Season'] <= 1995]['curr_pl_wins'] #tot_pl_games␣
 ↪tot_wins
#PJ_test_X = PJ_data[PJ_data['Season'] >= 1996].drop(columns = ['curr_pl_wins'])
#y_PJ_test = PJ_data[PJ_data['Season'] >= 1996]['curr_pl_wins'] #tot_pl_games␣
 ↪tot_wins

pipeline = make_pipeline(
    StandardScaler(),
    SGDRegressor()
```

```
)

#pipeline.fit(X = PJ_train_X, y = y_PJ_train)

RMSE = cross_val_score(pipeline,
                       X = PJ_data.drop(columns = ['curr_pl_wins']),
                       y = PJ_data['curr_pl_wins'],
                       cv = 3)  # cv refers to number of cross-validation cuts
MSE = RMSE*RMSE
print('Coach Phil Jackson data model:\n Mean of MSE\'s: ' + str(np.mean(MSE)))
print(' Median of MSE\'s: ' + str(np.median(MSE)) + '\n')


pipeline = make_pipeline(
    StandardScaler(),
    SGDRegressor(max_iter = 5000)
)

#pipeline.fit(X = GP_train_X, y = y_GP_train)

RMSE = cross_val_score(pipeline,
                       X = GP_data.drop(columns = ['curr_pl_wins']),
                       y = GP_data['curr_pl_wins'],
                       cv = 3)  # cv refers to number of cross-validation cuts
MSE = RMSE*RMSE
print('Coach Gregg Popovich data model:\n Mean of MSE\'s: ' + str(np.mean(MSE)))
print(' Median of MSE\'s: ' + str(np.median(MSE)) + '\n')


pipeline = make_pipeline(
    StandardScaler(),
    SGDRegressor(max_iter = 2000)
)

#pipeline.fit(X = PR_train_X, y = y_PR_train)

RMSE = cross_val_score(pipeline,
                       X = PR_data.drop(columns = ['curr_pl_wins']),
                       y = PR_data['curr_pl_wins'],
                       cv = 3)  # cv refers to number of cross-validation cuts
MSE = RMSE*RMSE
print('Coach Pat Riley data model:\n Mean of MSE\'s: ' + str(np.mean(MSE)))
print(' Median of MSE\'s: ' + str(np.median(MSE)) + '\n')
```

```
Coach Phil Jackson data model:
 Mean of MSE's: 0.42266762497053545
 Median of MSE's: 0.301994705906538
```

```
Coach Gregg Popovich data model:
 Mean of MSE's: 0.04069876419957324
 Median of MSE's: 0.04728929277416238

Coach Pat Riley data model:
 Mean of MSE's: 0.18686075449154757
 Median of MSE's: 0.11478822024023597
```

I wanted to look more specifically on the coaches, so the last block of code are individualized linear regression models for coaches Phil Jackson, Gregg Popovich, and Pat Riley. I decided to do cross-validation for all the models, and ran 3 fold cross validation for all 3 models. I decided to re use the same features from the second model that took the better features from the first model analyzing coefficients, except this time I included features tot_pl_wins and removed the columns of Coaches that were changed from categorical to one hot encoding which is no longer needed for analyzing a single coach. After running the models and looking at their MSE's, we can see the most accurate of the models is the one correlating to Gregg Popovich, having the smallest MSE. Pat Riley comes next, with Phil Jackson is last. Jackson may be last only because he had the least amount of seasons coached, but only fall 3 seasons short from the next least # of coach seasons.