# ERA predictor

May 29, 2020

## 1 Year to year ERA prediction

This notebook analyzes which statistics best predict a pitchers Earned Run Average, (ERA) from one year to the next

[347]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
import sklearn
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
from sklearn.model_selection import GridSearchCV

from xgboost import XGBRegressor

%matplotlib inline
```

We first import select pitching data for qualified starting pitchers from www.fangraphs.com in the date range 2002-2017, for an explanation of each statistic see https://www.fangraphs.com/library/pitching/ .

[348]:
```
stats=pd.read_csv('FanGraphs Leaderboard.csv')
```

[349]:
```
stats.head()
```

[349]:
```
   Season             Name     Team  Age    K/9  BB/9  K/BB   H/9  HR/9  WHIP  \
0    2015      Zack Greinke  Dodgers   31   8.08  1.62  5.00  5.98  0.57  0.84
1    2015      Jake Arrieta     Cubs   29   9.28  1.89  4.92  5.90  0.39  0.86
2    2014  Clayton Kershaw  Dodgers   26  10.85  1.41  7.71  6.31  0.41  0.86
3    2013  Clayton Kershaw  Dodgers   25   8.85  1.98  4.46  6.25  0.42  0.92
4    2005     Roger Clemens   Astros   42   7.88  2.64  2.98  6.43  0.47  1.01
```

```
   …  F-Strike%      K%     BB%    Soft%     Med%    Hard%    ERA   xFIP  SIERA  \
0  …     64.1 %  23.7 %  4.7 %  21.7 %  51.5 %  26.8 %  1.66  3.22   3.27
1  …     60.2 %  27.1 %  5.5 %  22.8 %  55.2 %  22.1 %  1.77  2.61   2.75
2  …     68.8 %  31.9 %  4.1 %  24.5 %  51.2 %  24.3 %  1.77  2.08   2.09
3  …     65.1 %  25.6 %  5.7 %  14.4 %  56.7 %  28.9 %  1.83  2.88   2.99
4  …     60.4 %  22.1 %  7.4 %  16.3 %  60.9 %  22.8 %  1.87  3.31   3.47

  playerid
0     1943
1     4153
2     2036
3     2036
4      815

[5 rows x 35 columns]
```

One sees that there is a column for each statistic and a row for each pitcher for each season

[350]:
```
stats.columns
```

[350]:
```
Index(['Season', 'Name', 'Team', 'Age', 'K/9', 'BB/9', 'K/BB', 'H/9', 'HR/9',
       'WHIP', 'BABIP', 'GB/FB', 'LD%', 'GB%', 'FB%', 'IFFB%', 'HR/FB',
       'O-Swing%', 'Z-Swing%', 'Swing%', 'O-Contact%', 'Z-Contact%',
       'Contact%', 'Zone%', 'SwStr%', 'F-Strike%', 'K%', 'BB%', 'Soft%',
       'Med%', 'Hard%', 'ERA', 'xFIP', 'SIERA', 'playerid'],
      dtype='object')
```

The included stats are listed above, I have only included rate stats because counting statistics will skew predicitions in favor of pitchers who have pitched more innings in a given season.

Below I reformat the data values in each column so that they are all floats and are suitable for analysis

[351]:
```
for col in stats.columns:
    if type(stats[col][1])==str:
        stats[col]=stats[col].str.replace('%','',regex=False)
```

[352]:
```
numbers=['Age', 'K/9', 'BB/9', 'K/BB', 'H/9', 'HR/9',
       'WHIP', 'BABIP', 'GB/FB', 'LD%', 'GB%', 'FB%', 'IFFB%', 'HR/FB',
       'O-Swing%', 'Z-Swing%', 'Swing%', 'O-Contact%', 'Z-Contact%',
       'Contact%', 'Zone%', 'SwStr%', 'F-Strike%', 'K%', 'BB%', 'Soft%',
       'Med%', 'Hard%', 'ERA', 'playerid','xFIP','SIERA']
for col in numbers:
    stats[col]=stats[col].astype(float)
```

In the next three cells I create a new dataframe from the original that includes a column for the given pitchers ERA the following season, and rename the columns accordingly

[353]:
```
predseasons=['stats'+str(i)+str(i+1) for i in range(2002,2017)]
i=2002
for season in predseasons:

    globals()[season] = pd.
 ↪merge(stats[stats['Season']==i],stats[stats['Season']==(i+1)][['Name','ERA']],on="Name")
    i+=1
```

[354]:
```
statsrel=pd.concat([globals()[season] for season in predseasons])
```

[355]:
```
statsrel=statsrel.rename(columns={'ERA_x':'ERA_current_year','ERA_y':
 ↪'ERA_next_year'})
```

[356]:
```
statsrel.head()
```

[356]:
```
   Season            Name       Team   Age    K/9  BB/9  K/BB   H/9  HR/9  \
0    2002  Pedro Martinez    Red Sox  30.0  10.79  1.81  5.98  6.50  0.59
1    2002      Derek Lowe    Red Sox  29.0   5.20  1.97  2.65  6.80  0.49
2    2002     Greg Maddux     Braves  36.0   5.33  2.03  2.62  8.76  0.63
3    2002      Barry Zito  Athletics  24.0   7.14  3.06  2.33  7.14  0.94
4    2002   Bartolo Colon      - - -  29.0   5.75  2.70  2.13  8.45  0.77

   WHIP  …   K%  BB%  Soft%  Med%  Hard%  ERA_current_year  xFIP  SIERA  \
0  0.92  …  30.4  5.1   13.8  67.6   18.7              2.26  2.61   2.43
1  0.97  …  14.9  5.6   13.3  71.2   15.4              2.58  3.42   3.18
2  1.20  …  14.4  5.5   15.6  66.8   17.6              2.62  3.61   3.75
3  1.13  …  19.4  8.3   18.7  61.4   19.9              2.75  4.28   4.13
4  1.24  …  15.4  7.3   17.4  61.1   21.5              2.93  4.06   4.26

   playerid  ERA_next_year
0     200.0           2.22
1     199.0           4.47
2     104.0           3.96
3     944.0           3.30
4     375.0           3.87

[5 rows x 36 columns]
```

[ ]:
```

```

"features" is a list of statistics that we will use to try to predict a pitchers ERA.

"featureswithestimators" also includes the ERA estimators xFIP (eXpected Fielding Independent Pitching) and SIERA (Skill Interactive ERA), these stats use strikeout, walk and fly ball rates to say what a pitchers ERA 'should' be by removing factors such as team defense that are out of the pitchers control. For more information on these stats see https://www.fangraphs.com/library/pitching/.

[357]:
```
features=['Age', 'K/9', 'BB/9', 'K/BB', 'H/9', 'HR/9',
       'WHIP', 'BABIP', 'GB/FB', 'LD%', 'GB%', 'FB%', 'IFFB%', 'HR/FB',
       'O-Swing%', 'Z-Swing%', 'Swing%', 'O-Contact%', 'Z-Contact%',
       'Contact%', 'Zone%', 'SwStr%', 'F-Strike%', 'K%', 'BB%', 'Soft%',
       'Med%', 'Hard%', 'ERA_current_year']
```

[358]:
```
featureswithestimators=['Age', 'K/9', 'BB/9', 'K/BB', 'H/9', 'HR/9',
       'WHIP', 'BABIP', 'GB/FB', 'LD%', 'GB%', 'FB%', 'IFFB%', 'HR/FB',
       'O-Swing%', 'Z-Swing%', 'Swing%', 'O-Contact%', 'Z-Contact%',
       'Contact%', 'Zone%', 'SwStr%', 'F-Strike%', 'K%', 'BB%', 'Soft%',
       'Med%', 'Hard%', 'ERA_current_year','xFIP','SIERA']
```

First lets explore which stats are most correlated with a pitcher's ERA during the same year

[359]:
```
corrdict1={col:np.abs(statsrel[col].corr(statsrel['ERA_current_year'])) for col␣
↪in featureswithestimators}






Corr=pd.DataFrame.from_dict(corrdict1, orient='index')
Corr.sort_values(by=0,ascending=False)
```

[359]:

| | 0 |
|---|---|
| ERA_current_year | 1.000000 |
| WHIP | 0.815086 |
| H/9 | 0.759634 |
| SIERA | 0.638081 |
| xFIP | 0.634663 |
| HR/9 | 0.590251 |
| K% | 0.534908 |
| K/BB | 0.489126 |
| K/9 | 0.452299 |
| SwStr% | 0.433424 |
| BABIP | 0.432780 |
| HR/FB | 0.423012 |
| Contact% | 0.411598 |
| O-Swing% | 0.353715 |
| Z-Contact% | 0.338963 |
| Soft% | 0.304414 |
| BB/9 | 0.299132 |
| F-Strike% | 0.248818 |
| Swing% | 0.233694 |
| Hard% | 0.230031 |
| BB% | 0.218658 |
| LD% | 0.144044 |
| O-Contact% | 0.125753 |

```
Zone%               0.114846
GB%                 0.107033
GB/FB               0.090136
IFFB%               0.080803
Z-Swing%            0.071524
Age                 0.064609
FB%                 0.059323
Med%                0.039181
```

We see that WHIP and hits per nine innings have strong correlations to ERA, placing just above xFIP and SIERA as the most strongly correlated variables. Now lets take a look at how these stats are correlated to a pitcher's ERA the following season.

[360]:
```
corrdict2={col:np.abs(statsrel[col].corr(statsrel['ERA_next_year'])) for col in␣
↪featureswithestimators}





Corr=pd.DataFrame.from_dict(corrdict2, orient='index')
Corr.sort_values(by=0,ascending=False)
```

[360]:

| | 0 |
|---|---|
| SIERA | 0.453758 |
| xFIP | 0.445884 |
| K% | 0.424969 |
| K/9 | 0.406644 |
| ERA_current_year | 0.339578 |
| K/BB | 0.335974 |
| SwStr% | 0.319018 |
| WHIP | 0.316859 |
| Contact% | 0.309912 |
| H/9 | 0.302348 |
| Z-Contact% | 0.290058 |
| HR/9 | 0.278060 |
| O-Swing% | 0.251271 |
| Soft% | 0.170777 |
| F-Strike% | 0.160567 |
| HR/FB | 0.156215 |
| Swing% | 0.145038 |
| Age | 0.117504 |
| Med% | 0.111272 |
| BB/9 | 0.106611 |
| Zone% | 0.103316 |
| BB% | 0.079882 |
| GB/FB | 0.056518 |
| GB% | 0.054717 |

```
O-Contact%        0.054508
Z-Swing%          0.049796
FB%               0.042389
LD%               0.038722
IFFB%             0.016310
Hard%             0.015875
BABIP             0.014046
```

Here we see a very different picture, SIERA and xFIP have moved to the front of the pack, justifying their use as truer measures of a pitchers talent than ERA alone. We see that WHIP, and H/9 are now only the 8th and 10th most correlated stats respectively.

Besides xFIP and SIERA, it seems that the best raw stats to use to predict a pitchers ERA are the ones involving strikeouts, (K%, K/9,K/BB, SwStr%). Its no surprise then that strikeouts are a major component of both the SIERA and xFIP estimators.

Also of note is that a pitchers walk rate, (BB%) has a very low correlation to his ERA the next year. It seems that walks do not hurt a pitcher very much as long as he maintains high strikeout totals.

Finally, we try to use a combination of the raw features in a linear regressor to predict a pitchers ERA the following year.

First we build our design matrix and define our feature scaler.

[422]:
```
scaler=StandardScaler()
```

[423]:
```
X = statsrel[features]
X.shape
```

[423]: (773, 29)

[424]:
```
y= statsrel['ERA_next_year'].values
len(y)
```

[424]: 773

Next we shuffle the order of the data so that the initial ordering does not bias the model.

[425]:
```
from sklearn.utils import shuffle
X,y=shuffle(X,y)
```

Now we can use a linear regressor on the data to try and make predictions. We test our model using 5 fold cross validation, and evaluate it using the root mean squared error.

[426]:
```
regressor = LinearRegression()
pipeline=Pipeline([('scaler',scaler),('regressor',regressor)])
MAE=cross_val_score(pipeline, X,y, cv=5,scoring='neg_mean_absolute_error').
 ↪mean()
-MAE
```

[426]: 0.5878540871270985

We see that our model is off by an average of about .588 runs in predicting a pitcher's ERA. For comparison, lets look at our errors if we use the pitchers ERA from the previous year as our predicition.

[427]:
```
MAEERA = sklearn.metrics.mean_absolute_error(y_true =␣
↪statsrel['ERA_next_year'], y_pred = statsrel['ERA_current_year'])
MAEERA
```

[427]: 0.7212031047865459

Our regressor fares significantly better, how about if we use SIERA and xFIP?

[428]:
```
MAESIERA = sklearn.metrics.mean_absolute_error(y_true =␣
↪statsrel['ERA_next_year'], y_pred = statsrel['SIERA'])
MAESIERA
```

[428]: 0.6183699870633894

[429]:
```
MAExFIP = sklearn.metrics.mean_absolute_error(y_true =␣
↪statsrel['ERA_next_year'], y_pred = statsrel['xFIP'])
MAExFIP
```

[429]: 0.6088874514877103

Our regressor fares slightly better than xFIP and SIERA, which is no surprise given they both largely use stats from our list of features as inputs.

We can improve our Regression model further by incorporating a regularization scheme, this will help keep the model from overfitiing the data. Lets first try a Ridge regressor, which helps keep the coefficients of our features small.

[430]:
```
from sklearn.linear_model import Ridge

regressor=Ridge(alpha=200)
pipeline=Pipeline([('scaler',scaler),('regressor',regressor)])

MAE=cross_val_score(pipeline, X,y, cv=5,scoring='neg_mean_absolute_error').
↪mean()
-MAE
```

[430]: 0.5809968690552829

We see a slight improvement in our MAE from earlier (.581 vs .589). Next lets try a Lasso regressor, this will make the algorithm concentrate on a smaller number of features.

[433]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Lasso
```

```
regressor=Lasso(alpha=.01)
pipeline=Pipeline([('scaler',scaler),('regressor',regressor)])

MAE=cross_val_score(pipeline, X,y, cv=5,scoring='neg_mean_squared_error').mean()
-MAE
```

[433]: 0.5155228569823673

Finally we try XGboost, a gradient boosted regressor:

[434]:
```
regressor = XGBRegressor(n_estimators=90,learning_rate=0.1,booster='gblinear')
pipeline=Pipeline([('scaler',scaler),('regressor',regressor)])
MAE=cross_val_score(pipeline, X,y, cv=5,scoring='neg_mean_squared_error').mean()
-MAE
```

[434]: 0.5170204185877213

In summary we find that Lasso Regression gives us the best estimator

One may wonder how I chose the specific parameters for my Regularization models, this is done using grid search. Grid search performs the regression task for a given parameter range and outputs the parameter that leads to the best performing model. An example of grid search for a Lasso Regression model is given below:

[435]:
```
regressor=Lasso()

pipeline=Pipeline([('scaler',scaler),('regressor',regressor)])
parameters = {'regressor__alpha': [0.005,0.01,0.02,0.05]}
model = GridSearchCV(pipeline,parameters, cv=5, iid=False,␣
 ↪n_jobs=-1,refit=True,verbose=5,scoring='neg_mean_absolute_error')
model.fit(X,y)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done   5 out of  20 | elapsed:    0.0s remaining:    0.1s
[Parallel(n_jobs=-1)]: Done  10 out of  20 | elapsed:    0.1s remaining:    0.1s
[Parallel(n_jobs=-1)]: Done  15 out of  20 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    0.1s finished
```

[435]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('scaler', StandardScaler(copy=True, with_mean=True,
with_std=True)), ('regressor', Lasso(alpha=1.0, copy_X=True, fit_intercept=True,
max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
```

```
    selection='cyclic', tol=0.0001, warm_start=False))]),
        fit_params=None, iid=False, n_jobs=-1,
        param_grid={'regressor__alpha': [0.005, 0.01, 0.02, 0.05]},
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring='neg_mean_absolute_error', verbose=5)
```

We print out the best parameter found using grid search and the associated croos-validation score

[436]:
```
print(model.best_params_)
print(-1*model.best_score_)
```

```
{'regressor__alpha': 0.02}
0.5825506882814334
```

Finally, lets look at the coefficients of our model to see which stats most strongly influence its prediction. (We use our best performing model, the Lasso regression)

[437]:
```
best_model=model.best_estimator_
```

[438]:
```
np.sort(np.abs(best_model.named_steps['regressor'].coef_))
```

[438]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.00256554, 0.00644565,
       0.00700854, 0.01604372, 0.03344088, 0.03958521, 0.04195595,
       0.06125372, 0.09552277, 0.11899635, 0.20287045])
```

Our top five coefficents are .203, .119, .096, .061, and .042 . Lets see which stats these correspond to

[439]:
```
most=np.argsort(np.abs(best_model.named_steps['regressor'].coef_))[-5:]

for i in reversed([statsrel[features].columns[i] for i in most]):
    print(i)
```

```
K%
HR/9
K/BB
Z-Contact%
O-Contact%
```

Given our correlation analysis above and the success of xFIP and SIERA it is unsurprising that K% and K/BB are among the strongest predictors of future performance. Interestingly, Z-Contact% (the rate at which a batter makes contact on pitches in the strike zone against a given pitcher) has a higher coefficient than we we would expect.