Baseball project documentation

Introduction :

In this project, we want to predict the performance of Major League Baseball (MLB) players in the future. We aim to factor in all the complex variables that may impact how a player will perform, and determine what their baseball statistics will be for a future date. In order to achieve our goal, we will use a decision tree. A decision tree is well suited to this problem because it can determine and apply rules in order of importance, and is extremely accurate and adaptable. Decision trees can easily grow with data and can also be easily combined with other techniques for even further accuracy.

Definition of the Problem:

Data mining can be used to solve many problems today. Available datasets such as baseball statistics over time can be data mined to obtain accurate predictions of how the data will look like in the future. Our problem is to use a large amount of baseball datasets over the span of five years in order to determine the performance of players in the future. This way, coaches, players and business investors can adequately prepare for the future and make the best decisions for their teams. Also, people placing bets in fantasy baseball games can optimize their bets so that they have the best chance to win.

Data Pre-processing Done:

The purpose of preprocessing is to convert raw data into a form that fits machine learning. Structured and clean data allows a data scientist to get more precise results from an applied machine learning model. The technique includes data formatting, cleaning, and sampling.

- Hardware and Software Requirements and Tools Used

- **Tools: Python 3.8.5, Jupyter Notebook, Numpy, Pandas, Matplotlib, Seaborn, Scikit-learn, Scipy**

- **Techniques: regression, Linear Regression ,Decision Tree Regressor, Lasso, Ridge, ElasticNet, SVR**
- **Hardware: I3 processor, 4GB RAM**

- Model/s Development and Evaluation

I have used the below models for regression:

1. Linear Regression
2. Decision Tree Regressor
3. Lasso
4. Ridge
5. ElasticNet
6. SVR

- Identification of possible problem-solving approaches (methods)

- Read the data (from csv)
- Identify the dependent and independent variables.
- Check if the data has missing values or the data is categorical or not.
- If yes, apply basic data preprocessing operations to bring the data in a go to go format.
- Now split the data into the groups of training and testing for the respective purpose.

- After splitting data, fit it to a most suitable model. (How to find a suitable model is answered below)
- Validate the model. If satisfactory, then go with it, else tune the parameters and keep testing. In a few cases, you can also try different algorithms for the same problem to understand the difference between the accuracies.
- From step 7 one can also learn about accuracy paradox.
- Visualize the data.

- Testing of Identified Approaches (Algorithms)

1. Cross validation
2. Linear Regression
3. Decision Tree Regression
4. r2_score
5. mean_squared_error, mean_absolute_error

- Run and Evaluate selected models

I have used the below models for regression:

Linear regression:

```
linreg = LinearRegression()
model = linreg.fit(x_train,y_train)
linreg.score(x_train,y_train)
y_pred = model.predict(x_test)
df_linreg_rmse = np.sqrt(mean_squared_error(y_test,y_pred))
df_linreg_rmse
```

```
4.024476115680575
```

Decision Tree Regressor:

```
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
dt.fit(x_train,y_train)
dt.score(x_train,y_train)
y_pred=dt.predict(x_test)
pred_train=dt.predict(x_train)
pred_test=dt.predict(x_test)
print(f"At random state {i},the traning accuracy is:- {r2_score(y_train,pred_train)}")
print(f"At random state {i},the testing accuracy is:- {r2_score(y_test,pred_test)}")
print("\n")
print('r2_score:',r2_score(y_test,y_pred))
print('Mean absolute error:',mean_absolute_error(y_test,y_pred))
print('Mean squared error:',mean_squared_error(y_test,y_pred))

print('Root Mean Squared Error:',np.sqrt(mean_squared_error(y_test,y_pred)))
```

```
At random state 99,the traning accuracy is:- 1.0
At random state 99,the testing accuracy is:- 0.528808693869182


r2_score: 0.528808693869182
Mean absolute error: 6.166666666666667
Mean squared error: 63.833333333333336
Root Mean Squared Error: 7.989576542804589
```

Lasso:

```
from sklearn.linear_model import Lasso,Ridge,ElasticNet
ls=Lasso(alpha=0.0001)
#ls=Lasso(alpha=1.0) # Default
ls.fit(x_train,y_train)
ls.score(x_train,y_train)
pred_train=ls.predict(x_train)
pred_test=ls.predict(x_test)
print(f"At random state {i},the traning accuracy is:- {r2_score(y_train,pred_train)}")
print(f"At random state {i},the testing accuracy is:- {r2_score(y_test,pred_test)}")
print("\n")
print('r2_score:',r2_score(y_train,pred_train))
print('Mean absolute error:',mean_absolute_error(y_train,pred_train))
print('Mean squared error:',mean_squared_error(y_train,pred_train))

print('Root Mean Squared Error:',np.sqrt(mean_squared_error(y_train,pred_train)))
```

```
At random state 99,the traning accuracy is:- 0.952801764490568
At random state 99,the testing accuracy is:- 0.8837763741957501


r2_score: 0.952801764490568
Mean absolute error: 1.6279299805187517
Mean squared error: 4.585767451475445
Root Mean Squared Error: 2.1414405085071695
```

Ridge:

```
rd=Ridge(alpha=0.0001)
#rd=Ridge()
rd.fit(x_train,y_train)
rd.score(x_train,y_train)
y_pred=rd.predict(x_test)
pred_train=rd.predict(x_train)
pred_test=rd.predict(x_test)
print(f"At random state {i},the traning accuracy is:- {r2_score(y_train,pred_train)}")
print(f"At random state {i},the testing accuracy is:- {r2_score(y_test,pred_test)}")
print("\n")
print('r2_score:',r2_score(y_test,y_pred))
print('Mean absolute error:',mean_absolute_error(y_test,y_pred))
print('Mean squared error:',mean_squared_error(y_test,y_pred))

print('Root Mean Squared Error:',np.sqrt(mean_squared_error(y_test,y_pred)))
```

```
At random state 99,the traning accuracy is:- 0.9531927759519933
At random state 99,the testing accuracy is:- 0.8810143815959045


r2_score: 0.8810143815959045
Mean absolute error: 3.615658714904358
Mean squared error: 16.11924613768817
Root Mean Squared Error: 4.014878097487913
```

ElasticNet:

```
enet = ElasticNet()
model = enet.fit(x_train,y_train)
enet.score(x_train,y_train)
y_pred = model.predict(x_test)
df_enet_rmse = np.sqrt(mean_squared_error(y_test,y_pred))
print('Root Mean Squared Error:',df_enet_rmse)
print('r2_score:',r2_scoresssSss(y_test,y_pred))
print('Mean absolute error:',mean_absolute_error(y_test,y_pred))
print('Mean squared error:',mean_squared_error(y_test,y_pred))
```

```
Root Mean Squared Error: 4.009745977347328
r2_score: 0.8813183799666371
Mean absolute error: 3.6889791974773516
Mean squared error: 16.07806280285308
```

SVR:

```
from sklearn.svm import SVR
svc=SVR()
svc.fit(x_train,y_train)
svc.score(x_train,y_train)

pred_train=svc.predict(x_train)
pred_test=svc.predict(x_test)
print(f"At random state {i},the traning accuracy is:- {r2_score(y_train,pred_train)}")
print(f"At random state {i},the testing accuracy is:- {r2_score(y_test,pred_test)}")
print("\n")
```

```
At random state 99,the traning accuracy is:- 0.0004338498665685808
At random state 99,the testing accuracy is:- -0.022529251933135486
```

- Conclusion:

The six models produced adequate results on their own, averaging a relatively low RMSE for each data set. Through these advanced algorithms and including pruning and other measures for over fitting, we were able to as a Lasso that performed pretty well on their own. This shows that these machine learning techniques can be applied well to predicting sports data. By ensembling multiple methods, we were able to create an even more accurate model. Lasso is the best model of this dataset, because

Lasso has better r2_score than linear regression model , Decision Tree model, ridge, Elastic Net for this dataset, r2_score is 95% it may also denote it is over fitting as it even classifies the outliers perfectly.