CS 513

Knowledge Discovery and Data Mining

Predict Career Longevity for NBA Rookies

Team Members

- Akshay Pradeep Patade - 20009092
- Deep Deven Manek - 20009220
- Feneel Doshi - 10476615
- Kevin Dsa - 20009000




About The Dataset

- The **National Basketball Association (NBA)** is a professional basketball league in North America. The league comprises 30 teams (29 in the United States and 1 in Canada) and is one of the four major professional sports leagues in the United States and Canada. It is the premier men's professional basketball league in the world.

- Dataset- https://www.kaggle.com/competitions/iust-nba-rookies/data

Problem Statement

- Career longevity is dependent on various factors for any player in all the games and so for NBA Rookies.
- The factors like games played, minutes played and other statistics like rebounds, assists, blocks, field goals is a major factor for any player during the game.
- We will be using various machine learning classification algorithms to predict where a NBA rookie will last more than 5 years in a league or not.

Data Fields

- **GP**: Games Played (here you might find some values in decimal, consider them to be the floor integer, for example, if the value is 12.789, the number of games played by the player is 12)
- **MIN**: Minutes Played
- **PTS**: Number of points per game
- **FGM**: Field goals made
- **FGA**: Field goals attempt
- **FG%**: field goals percent
- **3P Made**: 3 point made
- **3PA**: 3 points attempt
- **FTM**: Free throw made
- **FTA**: Free throw attempts
- **FT%**: Free throw percent
- **OREB**: Offensive rebounds
- **DREB**: Defensive rebounds
- **REB**: Rebounds
- **AST**: Assists
- **STL**: Steals
- **BLK**: Blocks
- **TOV:** Turnovers
- **3P%**: 3 point percent

Pre-Processing

We Performed EDA on the Data set

- We have 1101 Rows and 20 Columns in the dataset
- Since we do not have any missing values in the training and testing data set, Data Cleaning is not required

```
print("*"*30, "HEAD", "*"*30)
display(train.head(5))
print("*"*30, "SHAPE", "*"*30)
print(f"Rows: {train.shape[0]}\nColumns: {train.shape[1]}")
print("*"*30, "INFO", "*"*30)
display(train.info())
print("*"*30, "DESCRIBE", "*"*30)
display(train.describe().T)
print("*"*30, "NULL?", "*"*30)
display(train.isnull().sum())
print("*"*30, "EXPLAINING", "*"*30)
```

[49]

Pre-Processing

```
****************************** NULL? ******************************

GP          0
MIN         0
PTS         0
FGM         0
FGA         0
FG%         0
3P Made     0
3PA         0
3P%         0
FTM         0
FTA         0
FT%         0
OREB        0
DREB        0
REB         0
AST         0
STL         0
BLK         0
TOV         0
Target      0
dtype: int64
```

```
Data columns (total 20 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   GP       1101 non-null   float64
 1   MIN      1101 non-null   float64
 2   PTS      1101 non-null   float64
 3   FGM      1101 non-null   float64
 4   FGA      1101 non-null   float64
 5   FG%      1101 non-null   float64
 6   3P Made  1101 non-null   float64
 7   3PA      1101 non-null   float64
 8   3P%      1101 non-null   float64
 9   FTM      1101 non-null   float64
 10  FTA      1101 non-null   float64
 11  FT%      1101 non-null   float64
 12  OREB     1101 non-null   float64
 13  DREB     1101 non-null   float64
 14  REB      1101 non-null   float64
 15  AST      1101 non-null   float64
...
 18  TOV      1101 non-null   float64
 19  Target   1101 non-null   int64
dtypes: float64(19), int64(1)
```

Model Building

For this particular Dataset, Kaggle provides us with a specific **Train.csv** and **Test.csv** File

Hence there is no need to split the dataset

```
train = pd.read_csv("Train_data.csv")
test = pd.read_csv("Test_data.csv")
```

[47]

```
train.head()
```

[48]

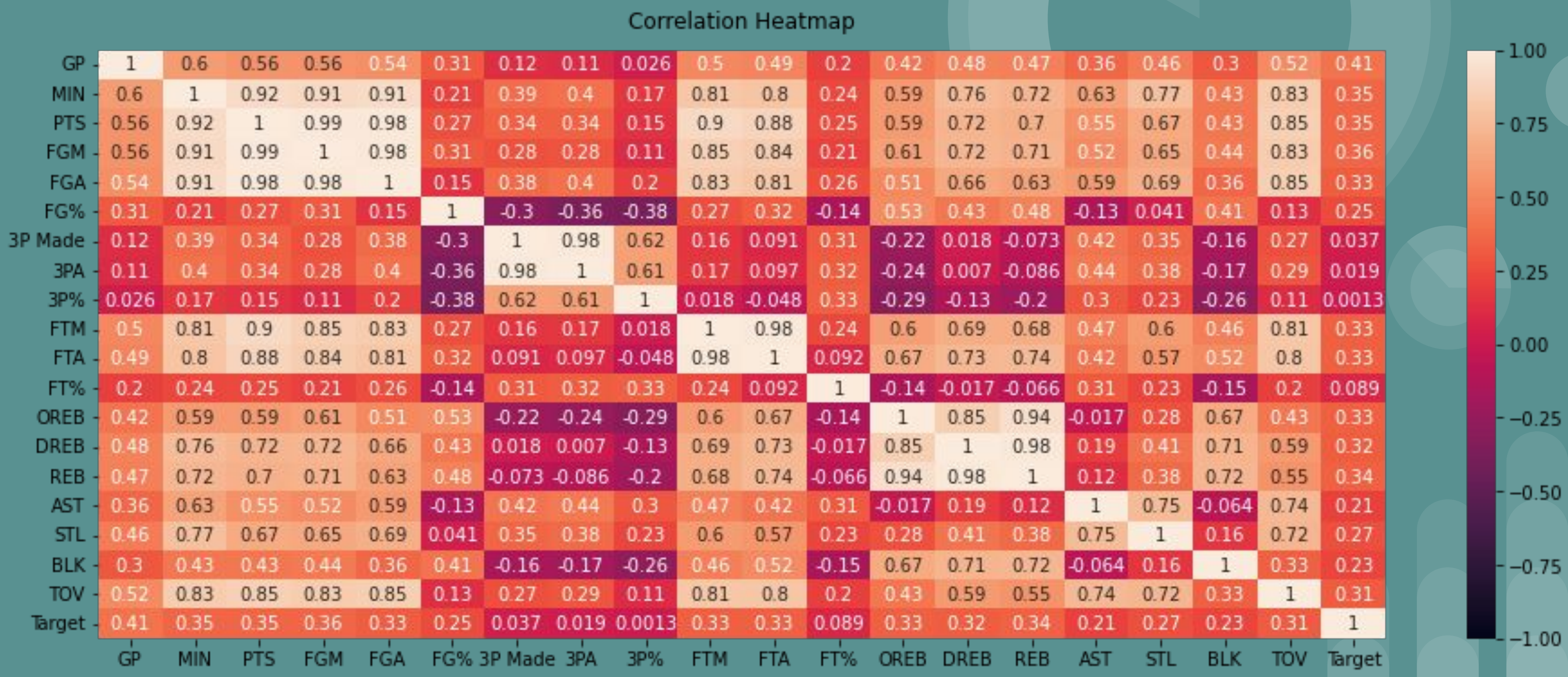
```
******************************* HEAD *******************************
```

	GP	MIN	PTS	FGM	FGA	FG%	3P Made	3PA	3P%	FTM	FTA	FT%	OREB	DREB	REB	AST	STL	BLK	TOV	Target
0	59.0	12.8	3.4	1.3	2.6	51.0	0.2	0.3	50.0	0.7	0.8	78.0	1.1	2.3	3.3	0.5	0.3	0.4	0.5	1
1	31.0	10.7	3.4	1.2	3.3	35.3	0.5	2.1	25.8	0.5	0.9	55.2	0.3	1.1	1.4	0.4	0.3	0.1	0.2	0
2	48.0	9.3	4.5	1.7	3.4	49.7	0.0	0.1	0.0	1.2	1.9	61.5	0.4	0.8	1.2	0.8	0.5	0.4	1.0	0
3	80.0	27.7	11.2	3.5	9.4	37.4	1.3	4.1	32.9	2.8	3.3	85.0	0.8	1.6	2.4	3.9	1.3	0.1	2.2	1
4	58.0	18.4	5.8	1.9	5.3	36.7	0.0	0.1	25.0	1.9	3.1	61.7	0.5	0.7	1.2	1.9	1.1	0.2	1.7	0

```
Output exceeds the size limit. Open the full output data in a text editor
******************************* SHAPE *******************************
Rows: 1101
Columns: 20
```

HeatMap

```python
plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(train.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);
```



Scaling

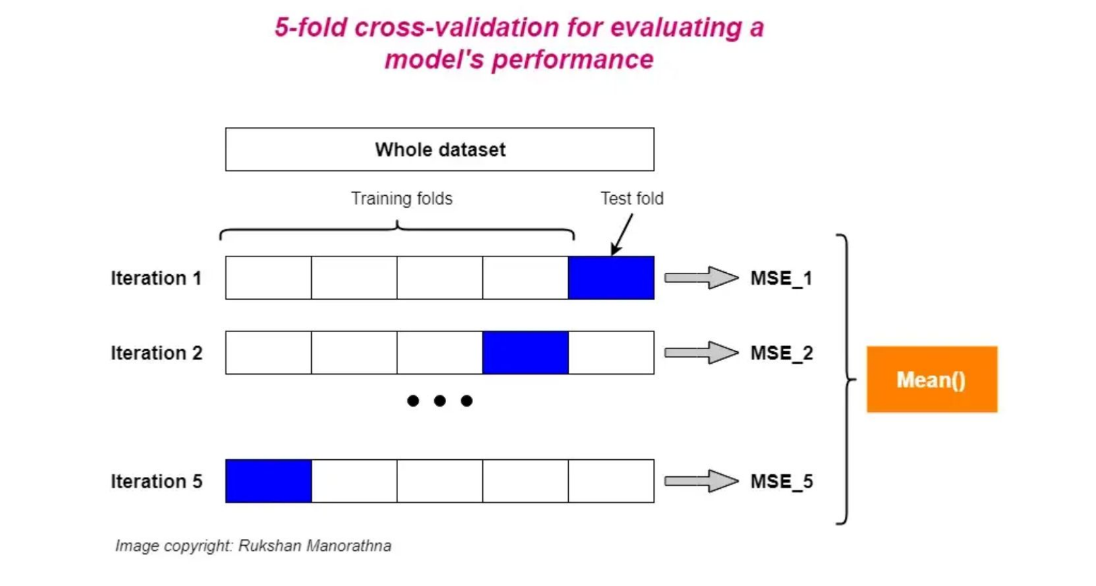
We are running our machine learning models on three types of data:

- Original data
- Normalized data (MinMax Scaler)
- Standardized data (Standard Scaler)

```python
X_train = X_train.copy()
X_test = X_test.copy()

#Standard Scaler
scaler = StandardScaler()
X_train_standard = scaler.fit_transform(X_train)
X_test_standard = scaler.fit_transform(X_test)

#Minmax Scaler
scaler = MinMaxScaler()
X_train_minmax = scaler.fit_transform(X_train)
X_test_minmax = scaler.fit_transform(X_test)


train_list = [X_train,X_train_standard,X_train_minmax]
scaler_list = ["without_scaler","standard_scaler","minmax_scaler"]
```

Additional Components

- We are using **K fold cross validation** to get the mean accuracy of each of the models.
- We are also using **GridSearchCV** to find the best parameters.



Classification Algorithms

- K - Nearest Neighbour
- Logistic Regression (lbfgs solver)
- Gaussian Naive Bayes
- Decision Tree
- Support Vector Machine
- Random Forest
- Random Forest with GridSearchCV
- Xgboost
- Xgboost with GridSearchCV

Code Snippet

```
random_state = 3
z = 0
for i in train_list:
    knn_model = KNeighborsClassifier().fit(i, y_train)
    logistic_model = LogisticRegression(solver='lbfgs', max_iter=400,random_state=random_state).fit(i,y_train)
    gaussian_model = GaussianNB().fit(i, y_train)
    decision_model = DecisionTreeClassifier(random_state=random_state).fit(i,y_train)
    linear_svm_model = SVC(kernel='linear').fit(i,y_train)
    randomforest_model = RandomForestClassifier(random_state=random_state).fit(i,y_train)

    #Random forest using grid search CV
    rfc = RandomForestClassifier(random_state = random_state)
    param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
    }

    CV_rfc = GridSearchCV(estimator = rfc, param_grid = param_grid, cv = 5)
    CV_rfc.fit(i, y_train)

    randomforest_cv_model = RandomForestClassifier(random_state = random_state, max_features = CV_rfc.best_params_.get("max_features"),
     n_estimators= CV_rfc.best_params_.get("n_estimators"), max_depth = CV_rfc.best_params_.get("max_depth"), criterion = CV_rfc.best_params_.get("criterion")).fit(i, y_train)
```

Code Snippet

```
#XG Boost using GridSeach CV
xgb = XGBClassifier(random_state = random_state)
grid_param = {
'n_estimators': [12, 25, 50, 75],
'max_depth': [3, 4, 5],
'learning_rate': [0.01, 0.05, 0.1]
}

#n_jobs = -1 means use all the processor
CV_xgb = GridSearchCV(estimator = xgb, param_grid = grid_param,
                      cv = 5, n_jobs = -1)

CV_xgb.fit(i, y_train)

xgb_cv_model = XGBClassifier(n_estimators = CV_xgb.best_params_.get("n_estimators"), max_depth = CV_xgb.best_params_.get("max_depth"),
learning_rate =  CV_xgb.best_params_.get("learning_rate")).fit(i, y_train)

model_names = ["Knn","Logistic","GaussianNB","DecisionTree","SupportVectorMachine","RandomForest","RandomForest Using GridSearch CV","Xgboost", "Xgboost Using GridSearch CV"]

model_list = [knn_model,logistic_model,gaussian_model,decision_model,linear_svm_model,randomforest_model,randomforest_cv_model,xgb_model,xgb_cv_model]
```

Code Snippet

```
results = []
z +=1
if z ==1:
    print("*"*30, f"{scaler_list[z-1]}","*"*30)
if z ==2:
    print("*"*30, f"{scaler_list[z-1]}","*"*30)
if z ==3:
    print("*"*30, f"{scaler_list[z-1]}","*"*30)
for j in model_list:
    result = cross_val_score(j, i, y_train, scoring = "accuracy", cv = 5, n_jobs=4)
    results.append(result.mean())

acc_of_models = {"Model": model_names, "Mean Accuracy": results}
acc_of_models = pd.DataFrame(acc_of_models)
display(acc_of_models)
```

Accuracy Comparison on Original Data

	Model	Mean Accuracy
0	Knn	0.702991
1	Logistic	0.699367
2	GaussianNB	0.673928
3	DecisionTree	0.653961
4	SupportVectorMachine	0.693920
5	RandomForest	0.743871
6	RandomForest Using GridSearch CV	0.745689
7	Xgboost	0.716635
8	Xgboost Using GridSearch CV	0.727511

Accuracy Comparison on Normalized Data

	Model	Mean Accuracy
0	Knn	0.690284
1	Logistic	0.694825
2	GaussianNB	0.673928
3	DecisionTree	0.653961
4	SupportVectorMachine	0.692110
5	RandomForest	0.745689
6	RandomForest Using GridSearch CV	0.743871
7	Xgboost	0.716635
8	Xgboost Using GridSearch CV	0.727511

Accuracy Comparison on Scaled Data

	Model	Mean Accuracy
0	Knn	0.670309
1	Logistic	0.694805
2	GaussianNB	0.673928
3	DecisionTree	0.654870
4	SupportVectorMachine	0.689371
5	RandomForest	0.742958
6	RandomForest Using GridSearch CV	0.742958
7	Xgboost	0.716635
8	Xgboost Using GridSearch CV	0.727511

Observations

1. Out of the nine Machine learning classification models **Random forest using Grid Search CV** has the highest mean accuracy of 74.29%.
2. Our Machine learning models interprets similar results for the Original, Normalized and Standardized dataset .

Thank You