K-NearestNeighbors-Copy1

August 20, 2021

[119]:
```
import sklearn.metrics
import sklearn.model_selection
import sklearn.neighbors
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
from scipy import stats
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
```

0.1 Can K-Nearest Neighbors Reveal Tipped Pitches?

[ ]:
```
from pybaseball import statcast_pitcher
degrom_stats = statcast_pitcher('2021-04-01', '2021-07-26', 594798)
```

For this mini project, I'm going to use pitching data from Major League Baseball. In Baseball, there is a concept called "tipping pitches", which means that some pitchers are prone to **unintentionally** telegraphing *which* type of pitch they are about to throw. Knowing what kind of pitch is coming can be invaluable information to a batter, as it can assist them in timing their swing better, or help them better decide if they want to swing at all. The best pitchers very rarely show any difference between their pitch types, as deception is key in getting batters out. Let's see if KNN can do better than the human eye and properly predict what pitch is coming based on some measured attributes prior to the release of the ball, namely: release position on an x axis, y axis, z axis, and release speed. Many other attributes exist in this dataset, but they might provide an unfair insight that would be impossible to react to in the moment, so they will be discarded.

0.1.1 Jacob DeGrom vs KNN:

One of the most dominant pitchers this 2021 season is Jacob DeGrom, a player for the New York Mets. DeGrom throws 4pitches classified by MLB's stat tracker:

[39]:
```
degrom_stats.pitch_type.unique()
```

[39]:
```
array(['FF', 'SL', 'CH', 'CU'], dtype=object)
```

- Changeup - (**CH**)
- Slider (**SL**)
- 4-Seam Fastball (**FF**)
- Curveball (**CU**)

Each of these pitches has very different movement patterns and speeds, but let's see if KNN can tell the difference based on their release point(x, y, z) and release speed.

0.1.2 Data Pre-Processing

As mentioned, many of the columns will be discarded. The data will also be scaled using MinMax. Again, our output variable of interest is **pitch_type**.

[40]:
```
# Select only relevant attributes
X = degrom_stats[['release_speed', 'release_pos_x', 'release_pos_y',␣
↪'release_pos_z']]
X.head(5)
```

[40]:
```
   release_speed  release_pos_x  release_pos_y  release_pos_z
0          100.2          -1.00          53.59           5.35
1           92.8          -1.04          53.66           5.47
2           99.4          -1.13          53.78           5.44
3           99.5          -1.15          53.78           5.43
4           98.7          -1.09          53.79           5.46
```

[44]:
```
# Scale features
mm_scale = MinMaxScaler()
X = mm_scale.fit_transform(X)
```

[50]:
```
# Make array of y values
y = np.ravel(degrom_stats[['pitch_type']].values)
y
```

[50]:
```
array(['FF', 'SL', 'FF', …, 'FF', 'FF', 'FF'], dtype=object)
```

Split Data

[53]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

0.1.3 Model Construction

Basic Model, untuned:

[58]:
```
knn_c = KNeighborsClassifier()
knn_c.fit(X_train, y_train)
print("Accuracy of untuned KNN Classifier", knn_c.score(X_test, y_test))
```

```
Accuracy of untuned KNN Classifier 0.9538043478260869
```

Amazing results! So, even one of the best pitchers in the world would fear a KNN-powered robot batter. Let's see if we can do better. #### Tuning K

[99]:
```
param_grid = {'weights': ['distance', 'uniform'], 'n_neighbors': [i for i in␣
↪range(1,15)]}
grid_search = GridSearchCV(knn_c, param_grid, scoring = "accuracy")
grid_search.fit(X, y)
grid_search.best_params_
```

```
/usr/local/lib/python3.9/site-packages/sklearn/model_selection/_split.py:666:
UserWarning: The least populated class in y has only 4 members, which is less
than n_splits=5.
  warnings.warn(("The least populated class in y has only %d"
```

[99]:
```
{'n_neighbors': 14, 'weights': 'uniform'}
```

We end up with changing the parameters to k = 14, and the weights remain uniform (as is the default).

[102]:
```
grid_search.best_estimator_.fit(X_train, y_train)
print("Test Accuracy:",grid_search.best_estimator_.score(X_test, y_test))
```

```
Test Accuracy: 0.9619565217391305
```

[103]:
```
print("Training Accuracy:", grid_search.best_estimator_.score(X_train, y_train))
```

```
Training Accuracy: 0.9603729603729604
```

0.1.4 Results

After the tuning, it looks like we were able to achieve just under a 1% increase in accuracy over the base model. Still, KNN has proved a valuable predictor of pitches. Perhaps KNN and other machine learning algorithms can assist batters in picking up on subtle physical queues that pitchers give. #### Overfitting The model doesn't overfit, as it generalizes well to the test set as well.