Pitch_Predictions

May 8, 2018

In [34]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression as logreg
from sklearn.neural_network import MLPClassifier as nn
from sklearn.neighbors import KNeighborsClassifier as knn
from sklearn.svm import SVC, LinearSVC
import matplotlib.pylab as plt
%matplotlib inline

pd.set_option('display.max_rows', 40)
pd.set_option('display.max_columns', 40)
```

In [3]:
```
# Retrieve 2 months of pitch data source
single_encoding = pd.read_csv("pitch_data_single_encoding_filter.csv")
# Add a column feature to see if the same handedness of pitcher and batter makes a diffe
single_encoding['diff_pit_bat_h'] = single_encoding.apply(lambda row: (int(row['bat_hand
# xor hands of pitcher and batter, 1 if different handed pitcher and batter, 0 if same
single_encoding.head()
```

In [4]:
```
# A correlation matrix was Calculated previously, and reduced the parameters going into
# to the following 22
pitchCategories = ['x', 'y', 'end_speed', 'diff_pit_bat_h',
    'pfx_x', 'pfx_z', 'px', 'pz', 'x0', 'y0', 'z0', 'vx0', 'vy0', 'vz0', 'ax',
    'ay', 'az', 'break_y', 'break_angle', 'break_length', 'spin_dir', 'spin_rate']

# Separates data into input parameters and corresponding labels
def separateData(pitchData):
    px = pitchData[pitchCategories].values
    py = pitchData[['pitch_event']].values[:, 0]

    return px, py
```

In [5]:
```
px, py = separateData(single_encoding)
```

In [7]:
```
# Gives the Decimal % Accuracy for a set of data
def getAccuracy(predict, actual):
    correct = 0
    if len(actual) != len(predict):
```

```
            raise ValueError('predict and actual arrays have different lengths')
        else:
            for i in range(len(actual) - 1):
                if predict[i] == actual[i]:
                    correct += 1

        return correct/len(predict)

    # Gives the counts classified for a set of data
    def getCounts(arr):
        occurences = {}
        for item in arr:
            if item in occurences:
                occurences[item] += 1
            else:
                occurences[item] = 1
        return occurences
```

In [8]:
```
# split the data into training and test
halfPt = round(len(py)/2)

# run logistic regression on the data
mdl2 = logreg()
mdl2.fit(px[:halfPt], py[:halfPt])
trainp = mdl2.predict(px[:halfPt])
testp = mdl2.predict(px[halfPt:])

traina = getAccuracy(trainp, py[:halfPt])
testa = getAccuracy(testp, py[halfPt:])

print('train accuracy', traina)
print('test accuracy', testa)


print('py_ array', getCounts(py))
print('train predictions', getCounts(trainp))
print('test predictions', getCounts(testp))
```

```
train accuracy 0.43644376036833654
test accuracy 0.43486341488297614
py_ array {2: 186737, 0: 403373, 3: 187438, 1: 124235}
train predictions {0: 443553, 3: 2901, 2: 4430, 1: 8}
test predictions {0: 443344, 3: 3027, 2: 4512, 1: 8}
```

In [9]:
```
# split the data into training and test
halfPt = round(len(py)/2)
```

```
        # run K-Nearest Neighbors on the data
        clf_ = knn(n_neighbors=50)
        clf_.fit(px[:halfPt], py[:halfPt])
        trainp = clf_.predict(px[:halfPt])
        testp = clf_.predict(px[halfPt:])

        traina = getAccuracy(trainp, py[:halfPt])
        testa = getAccuracy(testp, py[halfPt:])

        print('train accuracy', traina)
        print('test accuracy', testa)


        print('py_ array', getCounts(py))
        print('train predictions', getCounts(trainp))
        print('test predictions', getCounts(testp))

        #print(mdl.score(px_[100000:], py_[100000:]))
```

```
train accuracy 0.6166665188116001
test accuracy 0.620047860791189
py_ array {2: 186737, 0: 403373, 3: 187438, 1: 124235}
train predictions {0: 240870, 3: 86236, 2: 105520, 1: 18266}
test predictions {0: 240351, 2: 105748, 3: 86245, 1: 18547}
```

In [7]:
```
        # Attempted to run the data through, was incredibly slow and was only able to train on 5
        # samples (this took 35 minutes). The low accuracy result made me move away from this op
        halfPt = 50000 #round(len(py_2)/2)

        clf = SVC()
        clf.fit(px[:halfPt], py[:halfPt])
        trainp = clf.predict(px[:halfPt])
        testp = clf.predict(px[halfPt:])

        traina = getAccuracy(trainp, py[:halfPt])
        testa = getAccuracy(testp, py[halfPt:])

        print('train accuracy', traina)
        print('test accuracy', testa)

        print('py_ array', getCounts(py))
        print('train predictions', getCounts(trainp))
        print('test predictions', getCounts(testp))
```

```
train accuracy 0.99902
test accuracy 0.41518842215511015
py_ array {3: 290824, 2: 141831, 0: 309838, 1: 94251}
```

```
train predictions {3: 17362, 2: 8754, 0: 19058, 1: 4826}
test predictions {0: 748944, 3: 22870, 2: 9573, 1: 5357}
```

```
In [10]: # Ran the data through a Multi-Layer Perceptron Neural Network
         halfPt = round(len(py)/2)
         nnclf = nn()
         nnclf.fit(px[:halfPt], py[:halfPt])
         trainp = nnclf.predict(px[:halfPt])
         testp = nnclf.predict(px[halfPt:])

         traina = getAccuracy(trainp, py[:halfPt])
         testa = getAccuracy(testp, py[halfPt:])

         print('train accuracy', traina)
         print('test accuracy', testa)

         print('py_ array', getCounts(py))
         print('train predictions', getCounts(trainp))
         print('test predictions', getCounts(testp))
```

```
train accuracy 0.616100973182048
test accuracy 0.6164283607346344
py_ array {2: 186737, 0: 403373, 3: 187438, 1: 124235}
train predictions {3: 106000, 2: 106259, 0: 222316, 1: 16317}
test predictions {0: 222025, 2: 106486, 3: 105774, 1: 16606}
```

```
In [13]: # Imported the validation test data (2 days worth)
         test_pitches = pd.read_csv("pitch_test_data_filtered.csv")
         test_pitches['diff_pit_bat_h'] = test_pitches.apply(lambda row: (int(row['bat_hand_fl']
         test_pitches.head()
```

Out[13]:

	retro_game_id	inning	bat_home_id	pa_ball_ct	pa_strike_ct	outs_ct	\
0	CHN201708010	1	0	0	0	0	
1	CHN201708010	1	0	1	0	0	
2	CHN201708010	1	0	1	1	0	
3	CHN201708010	1	0	1	2	0	
4	CHN201708010	1	0	1	2	0	

	start_bases_cd	ab_number	x	y	end_speed	sz_top	sz_bot	\
0	0	1	173.42	208.82	85.1	3.501	1.664	
1	0	1	151.63	191.63	84.1	3.521	1.673	
2	0	1	164.24	206.29	83.9	3.316	1.397	
3	0	1	141.31	178.59	86.3	3.317	1.398	
4	0	1	119.29	188.36	83.8	3.224	1.305	

	pfx_x	pfx_z	px	pz	x0	y0	z0	vx0	vy0	vz0	\
0	6.847	8.763	-1.343	0.979	2.473	50	5.380	-12.461	-133.281	-8.591	

```
1  6.465   8.980 -0.796  1.656  2.456  50  5.409 -10.758 -132.578 -6.811
2  0.792   8.910 -1.084  1.082  2.571  50  5.439  -9.762 -130.467 -8.142
3  6.622   8.714 -0.487  2.152  2.499  50  5.380 -10.368 -136.050 -5.781
4  7.434  10.215  0.095  1.769  2.484  50  5.596  -8.753 -131.728 -7.278

       ax      ay      az  break_y  break_angle  break_length  spin_dir  \
0  12.493  24.807 -16.185     23.9        -30.1           4.5   141.999
1  11.617  25.811 -16.039     23.8        -29.0           4.5   144.249
2   1.394  22.161 -16.501     23.9          0.5           4.1   174.921
3  12.505  27.746 -15.718     23.8        -31.8           4.3   142.771
4  13.155  26.116 -14.097     23.8        -36.5           4.4   143.956

   spin_rate  pitch_event  pit_hand_fl  bat_hand_fl  diff_pit_bat_h
0   2208.653            0            1            1               0
1   2175.647            2            1            1               0
2   1749.724            1            1            1               0
3   2203.952            3            1            1               0
4   2462.278            3            1            1               0
```

In [33]:
```
#Separate data
px_, py_ = separateData(test_pitches)

# Run the validation data through the pretrained KNN model
testp = clf_.predict(px_)
testa = getAccuracy(testp, py_)

# Create a classification and correctness column for the validation dataframe
test_pitches['classification'] = testp
test_pitches['correct'] = test_pitches.apply(lambda row: (int(row['pitch_event']) == in

print('test accuracy', testa)
print('py_ array', getCounts(py_))
print('test predictions', getCounts(testp))

#Calculate the average middle of the strike zone, as well as the average range of it (f
avg_mid = (test_pitches['sz_top'].mean() + test_pitches['sz_bot'].mean())/2
print("average middle", avg_mid)
avg_ht_range = test_pitches['sz_top'].mean() - test_pitches['sz_bot'].mean()
print("average height range", avg_ht_range)

# Slim down the data to what will be exported to the app
test_pitches_slim = test_pitches[['retro_game_id', 'px', 'pz', 'sz_top', 'sz_bot', 'pit
print(test_pitches_slim.head())

# Split the dataframe to multiple dataframes grouped by game id
dfs = dict(tuple(test_pitches_slim.groupby('retro_game_id')))

# Export each game's dataframe of pitches to a separate csv
```

```
        for item in dfs:
            dfs[item].to_csv('webApp/src/app/test_game_data/test_data_' + item + '.csv', header
```

```
test accuracy 0.591870160810006
py_ array {0: 3004, 2: 1378, 1: 943, 3: 1391}
test predictions {0: 3611, 2: 1532, 3: 1263, 1: 310}
average middle 2.4679111078
average height range 1.84513609291
  retro_game_id     px     pz  sz_top  sz_bot  pitch_event  classification  \
0  CHN201708010 -1.343  0.979   3.501   1.664            0               0
1  CHN201708010 -0.796  1.656   3.521   1.673            2               0
2  CHN201708010 -1.084  1.082   3.316   1.397            1               0
3  CHN201708010 -0.487  2.152   3.317   1.398            3               2
4  CHN201708010  0.095  1.769   3.224   1.305            3               2

   correct
0     True
1    False
2    False
3    False
4    False
```