Building a prediction model for pitch type based on statistical variables.

I want to determine what the type of each pitch was during the game based on the pitching statistics provided. I will be using a decison tree to build this prediction model

First, let's load up the packages we will be using.

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        from pandas import DataFrame
        from scipy import stats
        %matplotlib inline
        import warnings
        warnings.filterwarnings('ignore')
        sns.set_context("notebook")  # make figures fit
        from pylab import rcParams
        import matplotlib.pyplot as plt
        plt.rcParams.update({'font.size': 19})
        sns.set(font_scale=1.5)
        plt.rcParams['figure.figsize'] = 7,4
        from statsmodels.graphics.mosaicplot import mosaic
        exec(open('useful.py').read())

        # make the Pandas tables a little more readable

        from IPython.core.display import HTML
        css = open('style-table.css').read() + open('style-notebook.css').read()
        HTML('<style>{}</style>'.format(css))
```

Now let's load the data.

```
In [42]: game_data = pd.read_csv("20170817.csv")
         game_data.shape
```

```
Out[42]: (296, 33)
```

We are given 296 rows and 33 columns that represent 296 pitches and the various statistics that happened on that pitch. If we want to determine what type of pitch a ball is, we need to think about which statistics might be important. Let's take a look at our columns.

```
In [43]: game_data.columns
```

```
Out[43]: Index(['pitchid', 'inning', 'inning_half', 'pitcher', 'throws', 'batter',
                'batside', 'catcher', 'bat_score', 'field_score', 'balls', 'strike
         s',
                'outs_before', 'outs_after', 'pitch_type', 'rel_speed', 'rel_heigh
         t',
                'rel_side', 'induced_vert_break', 'horz_break', 'zone_speed',
                'spin_rate', 'spin_axis', 'plate_loc_x', 'plate_loc_z', 'sz_top',
                'sz_bot', 'pitch_call', 'event_type', 'event_description',
                'exit_velocity', 'launch_angle', 'hit_direction'],
               dtype='object')
```

The speed of a pitch and the break would seem like very likely features for our training data. These fields are "rel_speed", "induced_vert_break", and "horz_break". Another key aspect would be spin. "Spin_axis" and "spin_rate" will be our last two features. Now that we know the data we are going to be working with, let's reduce our DataFrame down to the information we need.

```
In [44]: pitch_data = game_data[["pitch_type",'horz_break','induced_vert_break','spi
         pitch_data.head()
```

Out[44]:

	pitch_type	horz_break	induced_vert_break	spin_rate	spin_axis	rel_speed
0	FF	-7.41986	9.66393	2026.10	138.574	93.6440
1	FF	-7.03180	11.09360	1989.39	144.392	93.9607
2	FF	-9.73929	9.92929	2062.76	131.440	94.2005
3	FF	-10.38880	10.73850	1916.24	132.372	93.6269
4	FF	-15.36990	10.48860	1925.18	120.871	94.0730

We can do some simple statistics on these types of pitches with the groupby method. For example, let's find the average stats of these pitches, the maximum, and the minimum.

```
In [45]: pitch_data.groupby("pitch_type").mean()
```

Out[45]:

	horz_break	induced_vert_break	spin_rate	spin_axis	rel_speed
pitch_type					
CB	2.464201	-7.374823	2486.857632	197.376296	79.973189
CH	9.373023	7.984331	1762.692222	221.505637	85.502570
FC	3.208504	5.843544	2158.085000	215.903250	88.227762
FF	4.221474	13.493489	2263.438790	193.567624	95.169168
FS	-9.323395	6.350056	1494.237273	124.880000	84.381391
FT	15.471725	3.324565	2238.567500	262.535500	90.739925
SL	-1.864152	-1.231902	2167.117059	95.500563	87.664780

In [46]:

```
pitch_data.groupby("pitch_type").min()
```

Out[46]:

	horz_break	induced_vert_break	spin_rate	spin_axis	rel_speed
pitch_type					
CB	-16.168300	-12.08590	2261.19	5.69615	77.7298
CH	-17.288500	-1.99262	1375.68	93.19520	81.2822
FC	0.888103	3.14690	1977.64	192.06800	86.0980
FF	-15.980700	2.66932	1795.58	108.72500	88.8870
FS	-16.791300	4.47608	1422.90	103.56600	83.2412
FT	14.348100	1.27632	2214.54	257.85000	90.2411
SL	-7.337980	-8.90306	0.00	7.09555	82.6189

In [47]:

```
pitch_data.groupby("pitch_type").max()
```

Out[47]:

	horz_break	induced_vert_break	spin_rate	spin_axis	rel_speed
pitch_type					
CB	15.77190	-1.61767	2901.12	333.708	86.9156
CH	20.04730	14.99300	2313.97	282.262	89.7858
FC	5.41397	8.52567	2399.82	237.004	91.9617
FF	19.82670	20.35360	2645.64	261.843	99.5715
FS	19.21780	9.18782	1615.99	255.177	87.4298
FT	17.13570	4.45343	2258.31	270.386	91.2182
SL	6.38973	7.76618	3019.61	347.886	92.2858

Let's import our SKLearn module and our decision tree.

In [58]:

```
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
```

Now let's divide our data into the feature set & the target set. The “X” set will consist of predictor variables. It consists of data from 2nd column to 6th column. The “Y” set will consist of the outcome variable. It consists of data in the 1st column. We are using “.values” of numpy converting our dataframes into numpy arrays.

In [49]:

```
X = pitch_data.values[:, 1:6]
Y = pitch_data.values[:,0]
```

Let’s split our data into training and test set. We will use sklearn’s train_test_split() method. We will split data into a training and a test set. X_train, y_train are training data & X_test, y_test will belong to the test dataset.

The parameter test_size is a given value 0.3, meaning it will be 30% of whole dataset & training dataset’s size will be 70% of the entire dataset.

In [51]:

```
X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3,
```

DecisionTreeClassifier(): is the classifier function for DecisionTree. It is the main function for implementing the algorithms.

In [56]:

```

clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100, ma
clf_gini.fit(X_train, y_train)
clf_gini.score
```

Out[56]:

```
<bound method ClassifierMixin.score of DecisionTreeClassifier(class_weigh
t=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=5,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=100, splitter='best')>
```

Great. We've done it. We can now vizualize our decision tree. The code below will use a package called PyDotPlus to save a picture of the decision tree to my computer. We need to install it first.

In [77]:

```
import sys
!{sys.executable} -m pip install pydotplus
```

```
Collecting pydotplus
  Using cached pydotplus-2.0.2.tar.gz
Requirement already satisfied: pyparsing>=2.0.1 in /anaconda/envs/si370/l
ib/python3.5/site-packages (from pydotplus)
Building wheels for collected packages: pydotplus
  Running setup.py bdist_wheel for pydotplus ... - \ | done
  Stored in directory: /Users/Brian/Library/Caches/pip/wheels/43/31/48/e1
d60511537b50a8ec28b130566d2fbbe4ac302b0def4baa48
Successfully built pydotplus
Installing collected packages: pydotplus
Successfully installed pydotplus-2.0.2
You are using pip version 9.0.1, however version 9.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' comman
d.
```

In [79]:

```
import pydotplus
import collections
names = ['Horz_Break','Induced_Vert_Break','Spin_Rate','Spin_Axis',"Rel_Spee
# Visualize data
dot_data = tree.export_graphviz(clf_gini,
                                feature_names=names,
                                out_file=None,
                                filled=True,
                                rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)
colors = ('turquoise', 'orange')
edges = collections.defaultdict(list)

for edge in graph.get_edge_list():
    edges[edge.get_source()].append(int(edge.get_destination()))

for edge in edges:
    edges[edge].sort()
    for i in range(2):
        dest = graph.get_node(str(edges[edge][i]))[0]
        dest.set_fillcolor(colors[i])

graph.write_png('tree.png')
```

Out[79]:

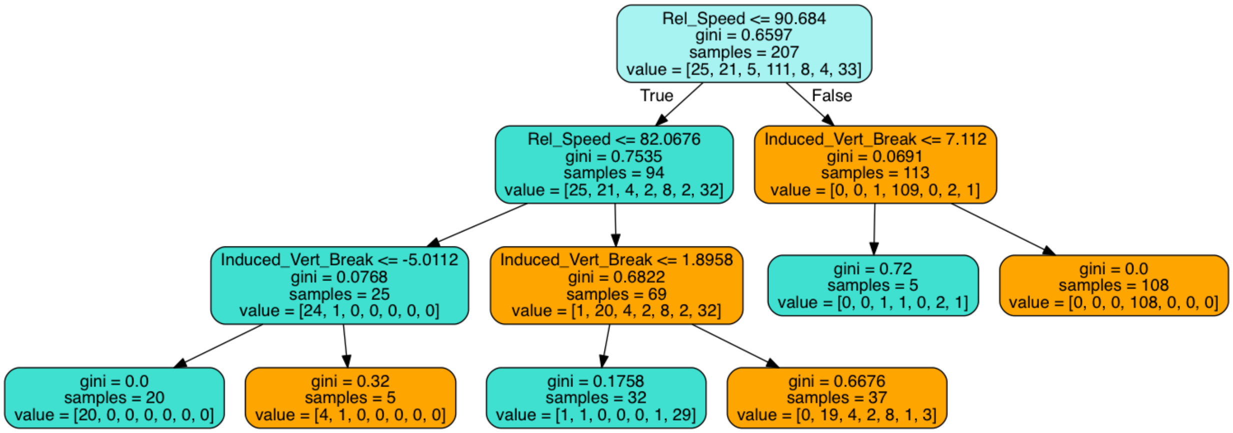
```
True
```

In [81]:

```

from IPython.display import Image
from IPython.core.display import HTML
Image(url= "tree.png")
```

Out[81]:


Rel_Speed <= 90.684
gini = 0.6597
samples = 207
value = [25, 21, 5, 111, 8, 4, 33]
True
False
Rel_Speed <= 82.0676
gini = 0.7535
samples = 94
value = [25, 21, 4, 2, 8, 2, 32]
Induced_Vert_Break <= 7.112
gini = 0.0691
samples = 113
value = [0, 0, 1, 109, 0, 2, 1]
Induced_Vert_Break <= -5.0112
gini = 0.0768
samples = 25
value = [24, 1, 0, 0, 0, 0, 0]
Induced_Vert_Break <= 1.8958
gini = 0.6822
samples = 69
value = [1, 20, 4, 2, 8, 2, 32]
gini = 0.72
samples = 5
value = [0, 0, 1, 1, 0, 2, 1]
gini = 0.0
samples = 108
value = [0, 0, 0, 108, 0, 0, 0]
gini = 0.0
samples = 20
value = [20, 0, 0, 0, 0, 0, 0]
gini = 0.32
samples = 5
value = [4, 1, 0, 0, 0, 0, 0]
gini = 0.1758
samples = 32
value = [1, 1, 0, 0, 0, 1, 29]
gini = 0.6676
samples = 37
value = [0, 19, 4, 2, 8, 1, 3]

We are ready to predict pitch types for our test set. We can use predict() method. Let's try to predict target variable for test set's 1st record.

```
In [62]: y_pred = clf_gini.predict(X_test)
         y_pred
```

```
Out[62]: array(['FF', 'FF', 'SL', 'FF', 'FF', 'FF', 'CH', 'CH', 'FF', 'FF', 'CH',
                'FF', 'CH', 'FF', 'SL', 'CB', 'FF', 'FF', 'FF', 'SL', 'CH', 'SL',
                'FF', 'FF', 'FF', 'FT', 'FF', 'CB', 'CH', 'FT', 'SL', 'SL', 'CB',
                'FF', 'CH', 'FF', 'FF', 'CB', 'SL', 'SL', 'FF', 'FF', 'CB', 'FF',
                'CH', 'CH', 'FF', 'FF', 'FF', 'FF', 'CH', 'FF', 'FF', 'CB', 'FF',
                'CB', 'SL', 'CB', 'CH', 'FF', 'CB', 'CH', 'CB', 'FT', 'SL', 'CH',
                'FF', 'FF', 'CH', 'FF', 'FF', 'FF', 'SL', 'FF', 'CH', 'SL', 'FF',
                'FF', 'CH', 'FT', 'SL', 'FF', 'SL', 'CB', 'FF', 'CB', 'FF', 'CH',
                'CH'], dtype=object)
```

Let's check the accuracy score of the descion tree using the accuray_score function.

```
In [65]: accuracy_score(y_test,y_pred)*100
```

```
Out[65]: 77.52808988764048
```

That's not bad. We can use our descion tree to predict pitch types based on the speed, curve, and break.. Let's try one more time! Let's try a typical curveball and use the following numbers:

Rel_Speed: 80

Spin_Axis: 200

Spin_Rate: 2500

Induced_Vertical_Break: -7

Horizontal Break: 2.5

The expected result should be a curveball (CB)!

```
In [69]: clf_gini.predict([2.5,-7,2500,200,80])
```

```
//anaconda/envs/si370/lib/python3.5/site-packages/sklearn/utils/validatio
n.py:395: DeprecationWarning: Passing 1d arrays as data is deprecated in
0.17 and will raise ValueError in 0.19. Reshape your data either using X.
reshape(-1, 1) if your data has a single feature or X.reshape(1, -1) if i
t contains a single sample.
  DeprecationWarning)
```

```
Out[69]: array(['CB'], dtype=object)
```

```
In [ ]:
```