3. MachineLearning

March 20, 2019

1 Machine Learning

With the NFL data I have, I want to be able to predict the SuperBowl Winner for the upcoming season. However, since my data relies on the season summary statistics, I would have to wait until the 2019 NFL regular season was completed to make actual predictions before the SuperBowl. Instead, I will make a prediction for the SuperBowl winner of this past season only using the first 9 years of the data.

```
In [1]: import pandas as pd
        %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        import matplotlib.patches as mpatches
        from sklearn.feature_extraction import DictVectorizer
        from sklearn.preprocessing import StandardScaler
        from sklearn.neighbors import KNeighborsRegressor
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.pipeline import Pipeline
        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import f1_score
        from sklearn.metrics import accuracy_score
        from sklearn.ensemble import RandomForestClassifier
```

```
In [2]: nfl = pd.read_csv("nfl.csv")
        nfl = nfl.set_index("Name")

        nfl_thru_2017 = nfl[nfl.Year != 2018]
        nfl_2018 = nfl[nfl.Year == 2018]
```

2 K-Nearest Neighbors

```
In [3]: X_train = nfl_thru_2017[["% 3rd Downs Converted", "% 4th Downs Converted", "3rd Downs A
                                "4th Downs Attempted", "Avg Yards Gained on 1st Down", "Fumble
                                "Total Points", "Yards per Play", "Completed Passes Greater th
                                "Pass Attempts per Game", "Pass Completion Percentage", "Passi
                                "Rush Attempts per Game", "Rush Yards per Carry", "Rushing TDs
```

```
X_val = nfl_2018[["% 3rd Downs Converted", "% 4th Downs Converted", "3rd Downs Attempte
                  "Avg Yards Gained on 1st Down", "Fumbles Lost", "Penalties", "Total
                  "Completed Passes Greater than 40 Yards", "Interceptions", "Pass Atte
                  "Pass Completion Percentage", "Passing TDs", "Sacks", "Rush Attempts
                  "Rushing TDs"]]

X_train_dict = X_train.to_dict(orient="records")
X_val_dict = X_val.to_dict(orient="records")

y_train = nfl_thru_2017["SuperBowl Winner"]
y_val = nfl_2018["SuperBowl Winner"]

vec = DictVectorizer(sparse=False)
vec.fit(X_train_dict)
X_train = vec.transform(X_train_dict)
X_val = vec.transform(X_val_dict)

scaler = StandardScaler()
scaler.fit(X_train)
X_train_sc = scaler.transform(X_train)
X_val_sc = scaler.transform(X_val)

best_k = []
best_train_accuracy = []
best_test_accuracy = []
best_train_f1_score = []
best_test_f1_score = []
best_train = 0
best_test = 0

for k in range(2,31): #5

    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train_sc, y_train)

    y_train_pred = model.predict(X_train_sc)
    y_val_pred = model.predict(X_val_sc)

    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_val, y_val_pred)
    train_f1 = f1_score(y_train, y_train_pred, average="micro")
    test_f1 = f1_score(y_val, y_val_pred, average="micro")

    if (train_f1 > best_train) & (test_f1 > best_test):
        best_train = train_f1
        best_test = test_f1
        best_k.append(k)
```

```
                best_train_accuracy.append(train_accuracy)
                best_test_accuracy.append(test_accuracy)
                best_train_f1_score.append(train_f1)
                best_test_f1_score.append(test_f1)

                my_y_train_pred = y_train_pred
                my_y_val_pred = y_val_pred


        print(best_k.pop(), "Nearest Neighbors Classification Model")
        print("Training Accuracy : ", best_train_accuracy.pop())
        print("Test Accuracy : ", best_test_accuracy.pop())
        print("Training F1 Score : ", best_train_f1_score.pop())
        print("Test F1 Score : ", best_test_f1_score.pop())
        print("")
        print("Training Prediction")
        print((pd.Series(my_y_train_pred).value_counts()/32).round())
        print("")
        print("Test Prediction")
        print(pd.Series(my_y_val_pred).value_counts().idxmax())
        print("")
        print("Actual Counts of Super Bowl Winners")
        print(nfl["SuperBowl Winner"].value_counts()/32)
```

```
2 Nearest Neighbors Classification Model
Training Accuracy :  0.597222222222
Test Accuracy :  0.28125
Training F1 Score :  0.597222222222
Test F1 Score :  0.28125

Training Prediction
New England Patriots    2.0
Baltimore Ravens        2.0
Denver Broncos          2.0
Green Bay Packers       1.0
New Orleans Saints      1.0
New York Giants         1.0
Philadelphia Eagles     0.0
Seattle Seahawks        0.0
dtype: float64

Test Prediction
Denver Broncos

Actual Counts of Super Bowl Winners
New England Patriots    3.0
Baltimore Ravens        1.0
New York Giants         1.0
```

```
Seattle Seahawks        1.0
Philadelphia Eagles     1.0
New Orleans Saints      1.0
Green Bay Packers       1.0
Denver Broncos          1.0
Name: SuperBowl Winner, dtype: float64
```

After assessing different values of k and different combinations of features, I have reached the conclusion that a 2 Nearest Neighbors Classification Model most accurately predicts SuperBowl Winners using the F1 score as a measure of "overall correctness". For each iteration of k's, if both the training F1 score and test F1 score were better than the previous, the current training F1 score and test F1 score became the "best".

However, from the results it appears that both the accuracy and F1 score are rather low (59% and 28%) and not ideal for picking a SuperBowl winners. This model overpredicted the amount of SuperBowls the Baltimore Ravens and Denver Broncos won and predicted the Philadelphia Eagles and Seattle Seahawks did not win, despite the fact they did win 1 SuperBowl each. Also, this model predicts that the Denver Broncos would have won the SuperBowl in 2019 when in fact the New England Patriots emerged victorious.

Because the K-Nearest Neighbors Classification Model is not the best, I will next use a Random Forest Classifier.

**Note that the Test Accuracy and Test F1 Score are represented as decimals because the SuperBowl Winner Label for each year has been assigned to every row in the DataFrame as a team name rather than binary. When this variable was represented as binary (1's for SuperBowl Winners and 0's for non-SuperBowl Winners), the model predicted there would not be a SuperBowl Winner. For this same reason, the value counts have been divided by 32 (number of NFL teams) so that the number of SuperBowls won will be correct.

3 Random Foresting

```
In [4]: best_i = []
        best_j = []
        best_train_accuracy = []
        best_test_accuracy = []
        best_train_f1_score = []
        best_test_f1_score = []
        best_train = 0
        best_test = 0

        for i in [10,11,12,14,15]:
            for j in range(10,1010,100):

                forest = RandomForestClassifier(max_depth=None, min_samples_split=i, n_estimat
                forest.fit(X_train_sc, y_train)

                y_train_pred = forest.predict(X_train_sc)
                y_val_pred = forest.predict(X_val_sc)
```

```
                train_accuracy = accuracy_score(y_train, y_train_pred)
                test_accuracy = accuracy_score(y_val, y_val_pred)
                train_f1 = f1_score(y_train, y_train_pred, average="micro")
                test_f1 = f1_score(y_val, y_val_pred, average="micro")

                if (train_f1 > best_train) & (test_f1 > best_test):
                    best_train = train_f1
                    best_test = test_f1
                    best_i.append(i)
                    best_j.append(j)
                    best_train_accuracy.append(train_accuracy)
                    best_test_accuracy.append(test_accuracy)
                    best_train_f1_score.append(train_f1)
                    best_test_f1_score.append(test_f1)

                    my_y_train_pred = y_train_pred
                    my_y_val_pred = y_val_pred



        print("Minimum Samples Split: ", best_i.pop(), "and ","N Estimators: ",best_j.pop())
        print("Training Accuracy : ", best_train_accuracy.pop())
        print("Test Accuracy : ", best_test_accuracy.pop())
        print("Training F1 Score : ", best_train_f1_score.pop())
        print("Test F1 Score : ", best_test_f1_score.pop())
        print("")
        print("Training Prediction: ")
        print((pd.Series(my_y_train_pred).value_counts()/32).round())
        print("")
        print("Test Prediction")
        print(pd.Series(my_y_val_pred).value_counts().idxmax())
        print("")
        print("Actual Counts of Super Bowl Winners")
        print(nfl["SuperBowl Winner"].value_counts()/32)
```

```
Minimum Samples Split:  10 and  N Estimators:  710
Training Accuracy :  0.996527777778
Test Accuracy :  0.875
Training F1 Score :  0.996527777778
Test F1 Score :  0.875

Training Prediction:
New England Patriots    2.0
New Orleans Saints      1.0
Baltimore Ravens        1.0
New York Giants         1.0
Seattle Seahawks        1.0
```

```
Philadelphia Eagles     1.0
Denver Broncos          1.0
Green Bay Packers       1.0
dtype: float64

Test Prediction
New England Patriots

Actual Counts of Super Bowl Winners
New England Patriots    3.0
Baltimore Ravens        1.0
New York Giants         1.0
Seattle Seahawks        1.0
Philadelphia Eagles     1.0
New Orleans Saints      1.0
Green Bay Packers       1.0
Denver Broncos          1.0
Name: SuperBowl Winner, dtype: float64
```

A Random Forest Classifier predicts the SuperBowl Winners much more accurately than the K-Nearest Neighbors Classification Model. We see that the Training Accuracy and Training F1 score are 99% ,and the Test Accuracy and Test F1 Score are 84%. These scores are 40% and 60% better than the scores obtained from the K-Nearest Neighbors Classification Model.

**Note that the similarly to above, the Test Accuracy and Test F1 Score are represented as decimals because the SuperBowl Winner Label for each year has been assigned to every row in the DataFrame as a team name rather than binary. When this variable was represented as binary (1's for SuperBowl Winners and 0's for non-SuperBowl Winners), the model predicted there would not be a SuperBowl Winner. For this same reason, the value counts have been divided by 32 (number of NFL teams) so that the number of SuperBowls won will be correct.

4 K-Means Clustering

Now, instead of predicting SuperBowl Winners, I want to explore how a K-Means Clustering Model will group this data. I will use 4 clusters because there are 32 NFL teams and therfore maybe, the model will create 8 teams per cluster per year.

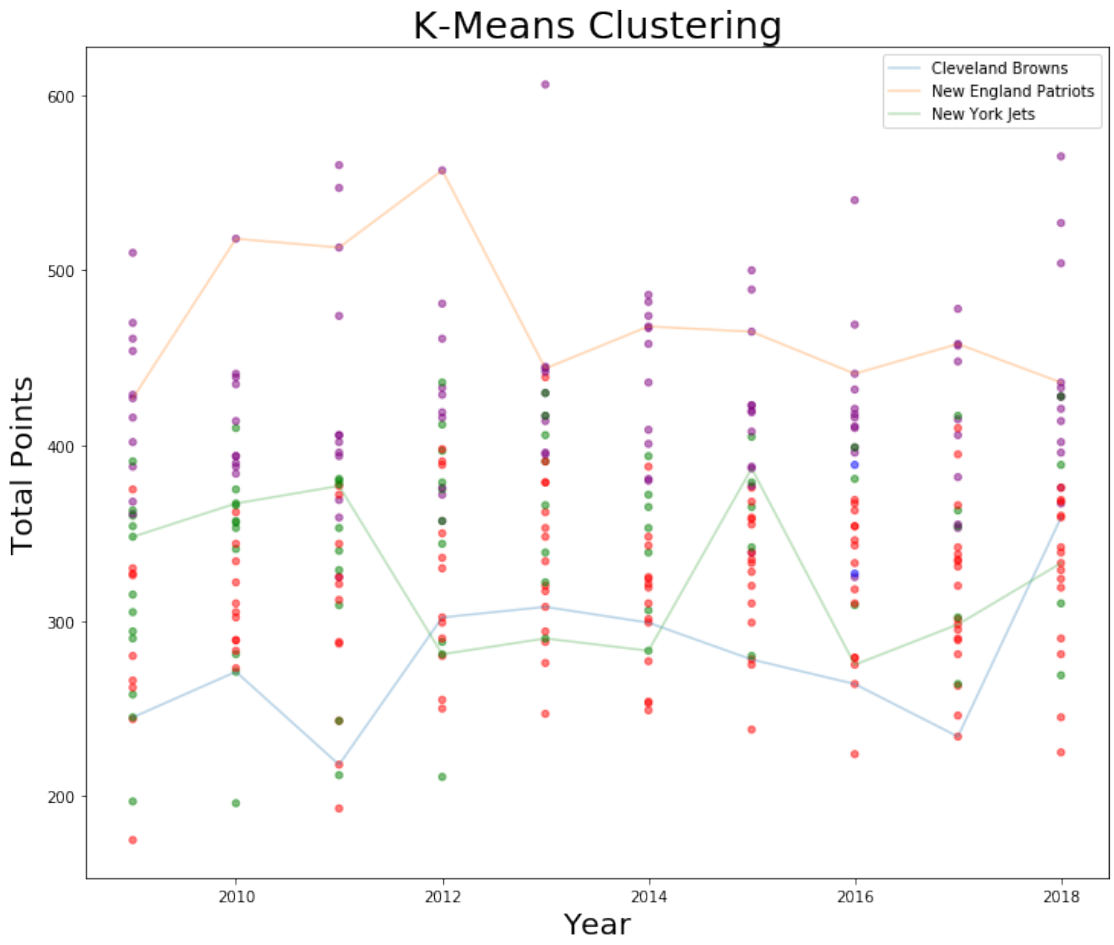
```
In [5]: from sklearn.cluster import KMeans

        df = nfl

        X_train_dict = df.to_dict(orient="records")

        vec = DictVectorizer(sparse=False)
        vec.fit(X_train_dict)
        X_train = vec.transform(X_train_dict)

        scaler = StandardScaler()
```

```
        scaler.fit(X_train)
        X_train_sc = scaler.transform(X_train)

        model = KMeans(n_clusters=4)
        model.fit(X_train_sc)

        centroids = model.cluster_centers_
        clusters = model.labels_

        clusters = pd.Series(clusters).map({
            0: "green",
            1: "blue",
            2: "red",
            3: "purple"
        })

        print("Cluster Predictions: ")
        print("")
        print(clusters.value_counts())
        print("")

        df.plot.scatter(x="Year", y="Total Points",
                              c=clusters, alpha=.5)


        fig = plt.gcf()
        fig.set_size_inches(12,10)
        plt.title("K-Means Clustering", size = 25)
        ax = plt.gca()
        ax.set_ylabel("Total Points", size = 20)
        ax.set_xlabel("Year", size = 20);

        cle = df.loc["Cleveland Browns"]
        plt.plot(cle["Year"], cle["Total Points"], alpha=.3, label="Cleveland Browns")

        ne = df.loc["New England Patriots"]
        plt.plot(ne["Year"], ne["Total Points"], alpha=.3, label="New England Patriots")

        nyj = df.loc["New York Jets"]
        plt.plot(nyj["Year"], nyj["Total Points"], alpha=.3, label="New York Jets")

        plt.legend()
```

```
Cluster Predictions:

red       135
purple    109
green      74
```

```
blue        2
dtype: int64
```

```
Out[5]: <matplotlib.legend.Legend at 0x7fe2e24ccf98>
```



Well it turns out that the value counts for each color are not equal. As a matter of fact, the value counts are all very different. Red seems to be most represented at the bottom of the graph, indicating this is most likely the worst teams from each season. Purple appears to mainly fill the upper part of the graph and therefore must be playoff teams. The few Green and Blue points that remain fill the gap between the Red and Purple points. As a check to confirm these colors correspond to good, average, and bad teams, I overlayed line graphs of 3 teams' Total Points. The yellow line graph at the top is the New England Patriots - a perennial playoff team and 3 time SuperBowl winning team during this timespan. The blue line graph at the bottom represents the Cleveland Browns, one of the worst teams this past decade. Lastly, this leaves the green line graph that strikes through the middle of the points in 2009-2011, 2015, and 2018. This line is the New York Jets, who have been a nearly .500 team the last 10 years. Therefore, based on the value counts and arrangement of the points, if we combine Blue clusters and Green Clusters, it appears that this NFL data is best mapped as 3 clusters total - good teams, average teams, and bad teams.