The-Music-Hall-Code

May 31, 2020

In [1]:
```
# setting python up with all the dependencies

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from imblearn.over_sampling import SMOTE
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score

# pointing python to where our data is, so it knows where to look

os.chdir('/Users/calgergen/Desktop/') # where data is stored on the computer
df = pd.read_csv('MusicHallData.csv') # reading in our dataset - has to be csv
```

```
Using TensorFlow backend.
```

In [2]:
```
df.fillna(df.mean(), inplace = True) # filling na's with column average
print(df['SuccessMetric'].value_counts()) # getting counts of each level of success me
# the print statement means that we'll see the results of this line
# after the code chunk runs

X = df.loc[:, df.columns != 'SuccessMetric'] # setting our predictor dataset - X
# Notice that X has all columns but our success metric
y = df['SuccessMetric'] # setting our response dataset - y
# Only has success metric problem

lab_enc = preprocessing.LabelEncoder()
y_enc = lab_enc.fit_transform(y)
y_df = pd.DataFrame(y_enc)
```

```
        y_df = y_df.rename(columns={0: 'SuccessMetric'})
        # this little section helps to format our response dataset so it is set to levels,
        # not a continuous variable

        X_smote = pd.DataFrame() # empty dataframe
        y_smote = pd.DataFrame() # empty dataframe

        for i in range(0, 100):
            sm = SMOTE(random_state = np.random.randint(0, 1000),
                       k_neighbors = 2) # smote setting
            X_res, y_res = sm.fit_sample(X, y_df) # sampling from X and y to create data
            X_res = pd.DataFrame(X_res, columns = X.columns)
            # giving created data original column names
            y_res = pd.DataFrame(y_res, columns = y_df.columns)
            X_smote = pd.concat([X_smote, X_res], axis=0)
            # attaching our created data to the empty dataframes we made earlier
            y_smote = pd.concat([y_smote, y_res], axis = 0)
```

```
9     23
10    22
8     13
7     13
5     12
6     10
3      4
4      3
Name: SuccessMetric, dtype: int64
```

```
In [3]: X_smote = X_smote.reset_index(drop = True) # resetting index for our new dataset
        y_smote = y_smote.reset_index(drop = True)
        final = X_smote.join(y_smote, how='outer') # joining our X and y created datasets
        print('Final Dataset Shape : ', final.shape) # Final shape of created dataset
```

```
Final Dataset Shape :  (18400, 25)
```

```
In [4]: print('Imbalance Check : ', final['SuccessMetric'].value_counts())
        # value counts for created success metric

        Xfinal = final.loc[:, final.columns != 'SuccessMetric'] # selecting only predictors
        yfinal = final['SuccessMetric'] # selecting only response

        X_train, X_test, y_train, y_test = train_test_split(
            Xfinal, yfinal, test_size=.2, random_state=0)
        # splitting our data into a train and test set

        rf = RandomForestRegressor(max_depth=5,
                                   n_estimators=100,
```

```
                                    max_features=10)
        # Random Forest setting

        rf.fit(X_train, y_train) # fitting our Random Forest model to our training data
        y_pred = rf.predict(X_test) # predicting outcomes based on our test data
        accuracy = r2_score(y_test, y_pred) # gathering our accuracy scores
        print('Accuracy with Train/Test Split of 20% : ', accuracy) # printing accuracy scores

        rf_model = rf.fit(X_train, y_train)
        col_99 = X.columns.values

        importances = list(rf_model.feature_importances_)
        feature_importances = [(feature,
                                round(importance, 2)) for feature,
                                importance in zip(col_99, importances)]
        feature_importances = sorted(feature_importances,
                                     key = lambda x: x[1], reverse = True)
        [print('Variable: {:20} Importance: {}'.
               format(*pair)) for pair in feature_importances];
        # this whole code chunk is working to give us our feature importance output
```

```
Imbalance Check :  7    2300
6    2300
5    2300
4    2300
3    2300
2    2300
1    2300
0    2300
Name: SuccessMetric, dtype: int64
Accuracy with Train/Test Split of 20% :  0.8592017059495108
Variable: MaxPrice_1820        Importance: 0.14
Variable: SpotifyPopularity    Importance: 0.1
Variable: AvgPrice_1820        Importance: 0.09
Variable: SpotifyMonthly       Importance: 0.07
Variable: PlaylistReachtoFollowersRatio Importance: 0.07
Variable: InstagramDNF         Importance: 0.07
Variable: PlaylistReachSpotify Importance: 0.06
Variable: SpotifyDNF           Importance: 0.04
Variable: FacebookDNLikes      Importance: 0.04
Variable: FacebookFans         Importance: 0.04
Variable: SEC_MinPrice_1820    Importance: 0.04
Variable: TotalYouTubeStreams  Importance: 0.03
Variable: FansDeezer           Importance: 0.03
Variable: TwitterDNF           Importance: 0.03
Variable: PercentSold_1820     Importance: 0.03
Variable: FanConversionRatio   Importance: 0.02
Variable: TwitterFollowers     Importance: 0.02
```

```
Variable: SEC_MaxPrice_1820    Importance: 0.02
Variable: Summer               Importance: 0.01
Variable: YouTubeSubscribers   Importance: 0.01
Variable: CPP                  Importance: 0.01
Variable: FansRanking          Importance: 0.01
Variable: MinPrice_1820        Importance: 0.01
Variable: SEC_AvgPrice_1820    Importance: 0.01
```

```
In [5]: test = pd.read_csv('MusicHallTestData.csv') # reading our csv of test artists
        artist = test['Artist'] # creating a variable for our artist so we can get it back lat
        test = test.drop('Artist', axis = 1) # dropping our artist variable
        test.fillna(df.mean(), inplace = True)
        # filling the na values in our test set the same way as earlier

        pred = rf.predict(test) # predicting outcomes for our test dataset
        rounder = [round(num) for num in list(pred)]
        # rounding final prediction to give final number

        test['Prediction'] = rounder # attaching predictions to dataset
        test['Artist'] = artist # reattaching artist to our dataset

        test.to_csv('Predictions.csv') # creating a final csv
        # this final csv will appear wherever you set python to look on your computer earlier

In [ ]:
```