Predicting Flight Delays Using US Department of Transportation's Data

MATH 2319 Machine Learning Applied Project Phase II

author: Manjushree Somashekar (s3712237) Krishna Sai Patha (s3670773)

date: 10 June 2018

Introduction

The objective of this project was to build Model to predict the flight delays at the take-off. The data sets were sourced from the Kaggle data (https://www.kaggle.com/usdot/flight-delays/data (https://www.kaggle.com/usdot/flight-delays/data)). This project has two phases. Phase I focuses on data preprocessing and exploration, as covered in Phase I report. We are presenting model building Phase II in this report. The rest of this report is organised as follow. Section 1 describes the flight data sets that were cleaned in Phase I process. Section 2 covers converting the Categorical values to matrices. Section 3, we split the data into test and training datasets. Section 4, Random forest search. Section 5, Plotting the histogram of the results and the last section ends with a summary.

The U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, canceled, and diverted flights is published in DOT's monthly Air Travel Consumer Report and in this dataset of 2015 flight delays and cancellations.
In 2015 there are lots of flight delays in United States for some reason. Nearly one third of all flights in our data set have delays. The main reasons for flight delays are wheather related but in some cases there are also airline or airport related flight delays.
In this pahase II, we aimed to examin and predict the delays and the cancellation causes in different aspects. We develop a model aimed at predicting flight delays at take-off. The purpose is not to obtain the best possible prediction but rather to emphasize on the various steps needed to build such a model

In [ ]:
```
import pandas as pd
import numpy as np
import os
import random
import scipy as sp
from sklearn.metrics import confusion_matrix

# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

In [7]:
```
flights = pd.read_csv('cleaned_flightsdata.cvs')
```

Convert Categorical values to matrices

Most of the values which is used for training the model are categorical and hence all the categorical values are converted to sparce matrices.

In [84]:
```
from sklearn.preprocessing import LabelBinarizer

binary_week = LabelBinarizer()
binary_week.fit(flights_orig['WEEK'])
final_binar_week = np.array(binary_week.transform(flights['WEEK']))
final_binar_week_orig = np.array(binary_week.transform(flights_orig['WEEK']))


day_binary_week = LabelBinarizer()
day_binary_week.fit(flights_orig['DAY_OF_WEEK'])
DAY_OF_final_binar_week = np.array(day_binary_week.transform(flights['DAY_OF_WEEK']))
DAY_OF_final_binar_week_orig = np.array(day_binary_week.transform(flights_orig['DAY_OF_WEEK']))


binar_airln = LabelBinarizer()
binar_airln.fit(flights_orig['AIRLINE'])
final_binar_airln = np.array(binar_airln.transform(flights['AIRLINE']))
final_binar_airln_orig = np.array(binar_airln.transform(flights_orig['AIRLINE']))


binar_orgAirpt = LabelBinarizer()
binar_orgAirpt.fit(flights_orig['ORIGIN_AIRPORT'])
final_binar_orgAirpt = np.array(day_binary_week.transform(flights['ORIGIN_AIRPORT']))
final_binar_orgAirpt_orig = np.array(day_binary_week.transform(flights_orig['ORIGIN_AIRPORT']))


binar_desAirpt = LabelBinarizer()
binar_desAirpt.fit(flights_orig['DESTINATION_AIRPORT'])
final_desAirpt = np.array(binar_desAirpt.transform(flights['DESTINATION_AIRPORT']))
final_desAirpt_orig = np.array(binar_desAirpt.transform(flights_orig['DESTINATION_AIRPORT']))


binar_schDep = LabelBinarizer()
binar_schDep.fit(flights_orig['SCHEDULED_DEPARTURE'])
final_binar_schDep = np.array(binar_schDep.transform(flights['SCHEDULED_DEPARTURE']))
final_binar_schDep_orig = np.array(binar_schDep.transform(flights_orig['SCHEDULED_DEPARTURE']))


# final_binar_testFlightData will contain all data in sparce matrix form
final_binar_testFlightData = np.concatenate((final_binar_week,DAY_OF_final_binar_week,final_binar_airln,final_binar_or
gAirpt,
```

```
                                      final_desAirpt,final_binar_schDep),axis=1)
final_binar_testFlightData_orig = np.concatenate((final_binar_week_orig,DAY_OF_final_binar_week_orig,final_binar_airln
_orig,final_binar_orgAirpt_orig,
                                      final_desAirpt_orig,final_binar_schDep_orig),axis=1)
```

Splitting dataset into test and training sets

In [85]:
```
# Splitting the data into test and train for which, the test data is 40% of split size
from sklearn.model_selection import train_test_split

trained_data, test_data, trained_data_target, test_data_target = train_test_split(final_binar_testFlightData, flights[
'late or cancelled'], test_size = 0.4, random_state=42)
```

Random Forest Search

In [86]:
```
from sklearn.ensemble import RandomForestRegressor
forstReg = RandomForestRegressor(n_jobs=-1)
forstReg.fit(trained_data,trained_data_target)
```

Out[86]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [87]:
```
confusion_matrix(trained_data_target,(forstReg.predict(trained_data)).round())
```

Out[87]:
```
array([[611,   7],
       [ 22, 618]])
```

In [88]:
```
confusion_matrix(test_data_target,(forstReg.predict(test_data)).round())
```

Out[88]:
```
array([[234, 167],
       [191, 247]])
```

In [89]:
```
#gridsearch to get the best conditions for random forest model
from sklearn.model_selection import GridSearchCV

grid_param = [{'n_estimators': [75, 100],'max_features':[10, 20],'max_depth':[20, 30]}]
forstReg = RandomForestRegressor(n_jobs=-1)
grid_search = GridSearchCV(forstReg, grid_param, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(trained_data,trained_data_target)
```

Out[89]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
           oob_score=False, random_state=None, verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'n_estimators': [75, 100], 'max_features': [10, 20], 'max_depth': [20, 30]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_squared_error', verbose=0)
```

Confusion matrix for train data

In [90]:
```
# confusion matrix for training set
confusion_matrix(trained_data_target,(grid_search.predict(trained_data)).round())
```

Out[90]:
```
array([[536,  82],
       [ 36, 604]])
```

In [91]:
```
grid_search.best_estimator_
```

Out[91]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=20,
           max_features=10, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=75, n_jobs=-1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

Confusion matrix for test data

In [92]:
```
# confusion matrix for test set
confusion_matrix(test_data_target,(grid_search.predict(test_data)).round())
```

Out[92]:
```
array([[215, 186],
       [147, 291]])
```

In [93]:
```
#Creating a column "Comparison", to show which results is true/false pos/neg
def comp_col(x):
    # True pos = 1, True neg = 2, False pos = 3, false neg = 4
    if (x['True'] == 1) & (x['predict'] > 0.5):
        return 1
    elif (x['True'] == 0) & (x['predict'] < 0.5):
        return 2
    elif (x['True'] == 0) & (x['predict'] > 0.5):
        return 3
    elif (x['True'] == 1) & (x['predict'] < 0.5):
        return 4
```

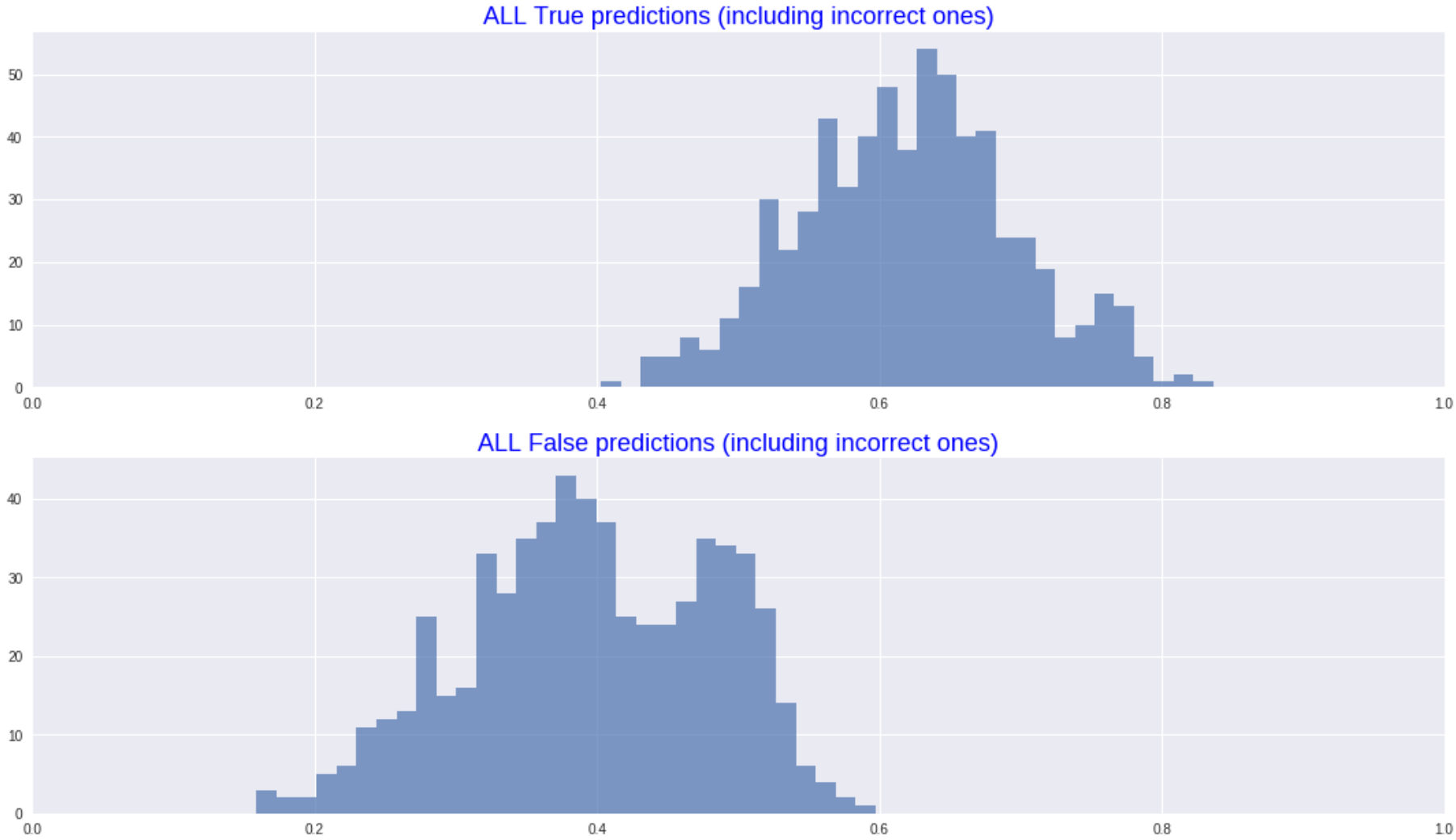
In [94]:
```
predictions = grid_search.predict(trained_data)
grid_reg_results = pd.DataFrame({'True':trained_data_target, 'predict':predictions})

grid_reg_results['Comprsn'] = grid_reg_results.apply(comp_col, axis = 1)
```

In [95]:
```
fig = plt.figure(figsize=(18.5, 10.5))
sub1 = fig.add_subplot(211) # instead of plt.subplot(2, 2, 1)
sub1.set_title('ALL True predictions (including incorrect ones)', fontsize=18, color="blue")
sub1.hist(grid_reg_results[grid_reg_results['True']==1]['predict'],alpha=0.7, bins=31)
sub1.set_xlim([0, 1])

sub2 = fig.add_subplot(212)
sub2.set_title('ALL False predictions (including incorrect ones)', fontsize=18, color="blue")
sub2.hist(grid_reg_results[grid_reg_results['True']==0]['predict'],alpha=0.7, bins=31)
sub2.set_xlim([0, 1])
```

```
Out[95]: (0, 1)
```


ALL True predictions (including incorrect ones)
0
10
20
30
40
50
0.0
0.2
0.4
0.6
0.8
1.0
ALL False predictions (including incorrect ones)
0
10
20
30
40
0.0
0.2
0.4
0.6
0.8
1.0

In [96]:
```
plt.hist(grid_reg_results[grid_reg_results['Comprsn'].isin([1,2])]['predict'],alpha=0.7, bins=30,label='Correct',color='c')
plt.hist(grid_reg_results[grid_reg_results['Comprsn'].isin([3,4])]['predict'],alpha=0.7, bins=30,label='Incorrect',color='m')
plt.legend(loc='upper left')
plt.title('Training Data')
```

Out[96]: Text(0.5,1,u'Training Data')

In [97]:
```
test_predictions = grid_search.predict(test_data)
test_grid_reg_results = pd.DataFrame({'True':test_data_target, 'predict':test_predictions})
test_grid_reg_results['Comprsn'] = test_grid_reg_results.apply(comp_col, axis = 1)
```

In [98]:
```
#Histograms for results
plt.hist(test_grid_reg_results[test_grid_reg_results['Comprsn'].isin([1,2])]['predict'],alpha=0.7, bins=30,label='Corr
ect',color='c')
plt.hist(test_grid_reg_results[test_grid_reg_results['Comprsn'].isin([3,4])]['predict'],alpha=0.7, bins=30,label='Inco
rrect',color='m')
plt.legend(loc='upper left')
plt.title('Test Data')
```

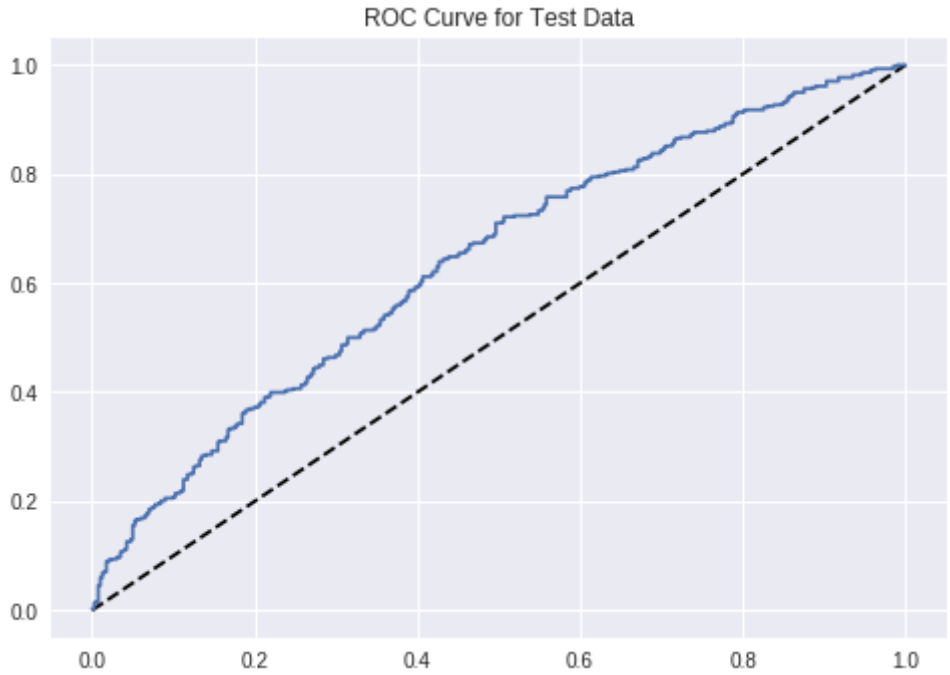
Out[98]: Text(0.5,1,u'Test Data')

In [99]:
```
from sklearn.metrics import roc_curve, auc

fpr_rf, tpr_rf, _ = roc_curve(test_data_target, test_predictions)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.title('ROC Curve for Test Data')
```

Out[99]: Text(0.5,1,u'ROC Curve for Test Data')



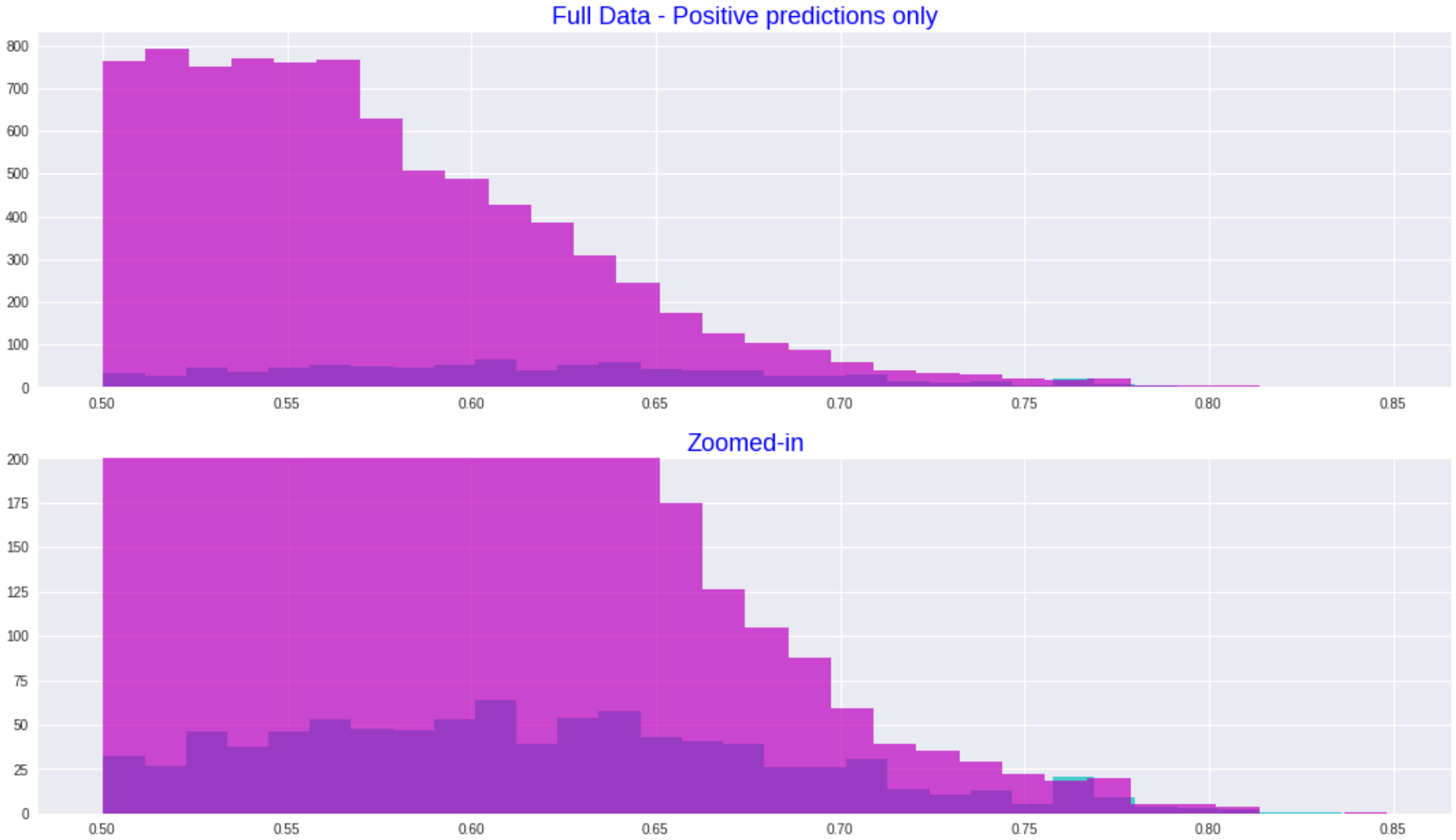
Modifying the final_binar_testFlightData_orig data before balancing late vs. not_late

In [100]:
```
orig_predictions = grid_search.predict(final_binar_testFlightData_orig)
orig_target = flights_orig['late or cancelled']
orig_grid_reg_results = pd.DataFrame({'True':orig_target, 'predict':orig_predictions})
orig_grid_reg_results['Comprsn'] = orig_grid_reg_results.apply(comp_col, axis = 1)
```

In [101]:
```
fig = plt.figure(figsize=(18.5, 10.5))
sub1 = fig.add_subplot(211) # instead of plt.subplot(2, 2, 1)
sub1.set_title('Full Data - Positive predictions only', fontsize=18, color="blue")
sub1.hist(orig_grid_reg_results[orig_grid_reg_results['Comprsn'].isin([1])]['predict'],alpha=0.7, bins=30,label='True
 Positive',color='c')
sub1.hist(orig_grid_reg_results[orig_grid_reg_results['Comprsn'].isin([3])]['predict'],alpha=0.7, bins=30,label='False
 Positive',color='m')
#sub1.set_xlim([0, 1])

sub2 = fig.add_subplot(212)
sub2.set_title('Zoomed-in', fontsize=18, color="blue")
sub2.hist(orig_grid_reg_results[orig_grid_reg_results['Comprsn'].isin([1])]['predict'],alpha=0.7, bins=30,label='True
 Positive',color='c')
sub2.hist(orig_grid_reg_results[orig_grid_reg_results['Comprsn'].isin([3])]['predict'],alpha=0.7, bins=30,label='False
 Positive',color='m')
sub2.set_ylim([0, 200])
#sub1.set_xlim([0, 1])
```

Out[101]: (0, 200)



In [102]:
```
confusion_matrix(orig_target,(orig_predictions).round())
```

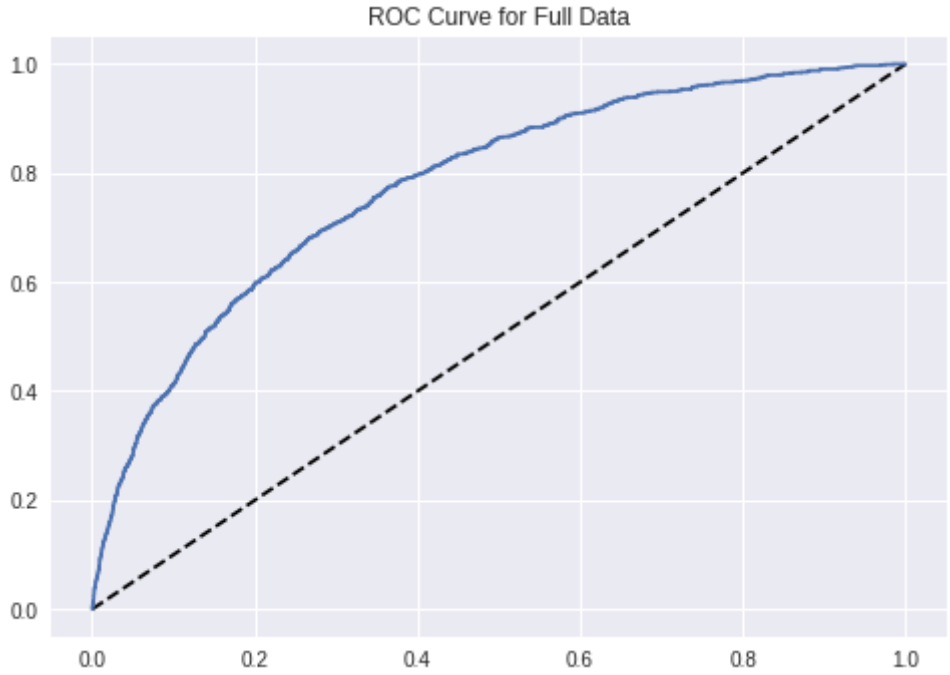
Out[102]:
```
array([[10280,  8328],
       [  183,   895]])
```

In [103]:
```
from sklearn.metrics import roc_curve, auc

fpr_rf, tpr_rf, _ = roc_curve(orig_target, orig_predictions)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.title('ROC Curve for Full Data')
```

Out[103]: Text(0.5,1,u'ROC Curve for Full Data')



Conclusion

Our initial plan was that by adding in weather data to the initial US Department of Transportation dataset provided by Kaggle, we could better predict whether a flight would be delayed or not, both across all airports as well as more granularly at an individual airport. Based on the ROC curves provided above, we do see that our best model outperforms that same model without the weather data set. This project also demonstrated the challenges of using machine learning methods to classify on imbalanced datasets. We attempted to correct for feeding more majority class data into the models by balancing the dataset but found that this led to models over-predicting the chance of delay on actual test sets. Therefore, we elected to keep an unbalanced data set but to look at multiple performance metrics including confusion matrices and ROC curves rather than simple accuracy metrics in order to better evaluate our models. We found that our initial best models chosen by optimizing deviance may not necessarily be the best ones to use given the ROC curve results.

The Random Forest produces the best performance in predicting individuals. We split the data into training and test sets. Observing the ROC, which looks quite good. When we observe the histogram of the whole data, there was much higher number of the instance where the flights were neither delayed by an hour or more but not cancelled(as we expected). It is quite hard to make a good or accurate predictions, which can be suspected from the confusion matrix.

But this model can be useful to alert passengers regarding their flights being delayed to cancelled by using predictions with the score higher than 85% (0.85)

Reference

https://www.kaggle.com/fabiendaniel/predicting-flight-delays-tutorial (https://www.kaggle.com/fabiendaniel/predicting-flight-delays-tutorial)
https://www.kaggle.com/dongxu027/airline-delays-eda-deep-divelessons-learned/notebook (https://www.kaggle.com/dongxu027/airline-delays-eda-deep-dive-lessons-learned/notebook")
https://www.transtats.bts.gov/HomeDrillChart.asp (https://www.transtats.bts.gov/HomeDrillChart.asp)
https://arxiv.org/pdf/1703.06118.pdf (https://arxiv.org/pdf/1703.06118.pdf)
https://www.transportation.gov/individuals/aviation-consumer-protection/flight-delays-cancellations (https://www.transportation.gov/individuals/aviation-consumer-protection/flight-delays-cancellations)
https://www.transportation.gov/airconsumer/flight-delays (https://www.transportation.gov/airconsumer/flight-delays)