project

January 17, 2022

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as any collaborators you worked with:

[2]:
```
COLLABORATORS = "Ruoxi Li"
# Note: In this project, my teammate Ruoxi Li and I discussed the theme,␣
↪development, content and conclusion of the project.
# We were sitting together and working together, which meant that each of us␣
↪contributed equally to the project.
# Also each section of this project is equally contributed.
```

---

[1]:
```
%matplotlib inline
%precision 16
import numpy
import matplotlib.pyplot as plt
import pandas as pd
```

1 Final Project

This notebook will provide a brief structure and rubric for presenting your final project.

The purpose of the project is 2-fold * To give you an opportunity to work on a problem you are truly interested in (as this is the best way to actually learn something) * To demonstrate to me that you understand the overall workflow of problem solving from problem selection to implementation to discussion

You can choose any subject area that interests you as long as there is a computational component to it. However, please do not reuse projects or homeworks you have done in other classes. This should be **your** original work.

You can work in teams, but clearly identify each persons contribution and every team member should hand in their own copy of the notebook.

1.0.1 Structure

There are 5 parts for a total of 100 points that provide the overall structure of a mini research project.

- Problem Description
- Problem Justification
- Description of Computational components needed to address problem
- Implementation including tests
- Discussion of results and future directions

For grading purposes, please try to make this notebook entirely self contained.

The project is worth about 2 problem sets and should be of comparable length or less (please: I will have about 100 of these to read and I am not expecting full 10 page papers). The actual project does not necessarily have to work but in that case you should demonstrate that you understand why it did not work and what steps you would take next to fix it.

Have fun

1.1 Problem Description [15 pts]

In 2-4 paragraphs, describe the general problem you want to solve and the goals you hope to achieve. You should provide any relevant background and references, particularly if you are reproducing results from a paper. Please use proper spelling and grammar.

Till now, flight delay has been studied mainly from a binary and regression standpoint. In this project, we will use random forest, a very popular classification method in machine learning, to predict whether the plane will be late. In this project, we consider using the random forest regression to predict the arrival delay in this dataset. Since the data set is big enough and has several variables(predictors), it's a good way to try to use the random forest method to do the prediction. In this project, we split the training and testing data. Also, our dataset does not have the problem of covariant translation since we have most of the time period in this data set. We first found a CSV data set called PNWFlights14, which is very large, with a lot of observation data and a lot of variables. Later we found a weather data set on the Internet that fits nicely with the airplane. Intuitively we think that the weather condition would cause the arrival delay, in other words, the weather condition can be a perfect predictor(variables) to fit a classification model to predict the arr_delay. And then we combined those two data sets, so we have both flight data and weather data for that day. Hopefully, this will make our model more accurate.

Our main steps in the implementation section are:

1. Get the two dataset and merge them together. Understand what each column stands for.

2. Do the data cleaning, delete non-sense rows of the entire data set. Based on the definition of each column, check the quality of the data set using numpy.testing.

3. Do some variable visualization.

4. Use the random forest method to make the model and do the prediction.

5. Calculate the model score, AUC score, confusion matrix, plot the ROC curve and test the effect of the model.

1.2 Problem Justification [5 pts]

Briefly describe why this problem is important to you, and, if possible, to anyone else.

As the most convenient long-distance transportation means, it is very important for us to know whether the plane will be late in advance. At the same time, for the airport, to ensure the continued operation of the airport, the operating schedule needs to cope with these flight delays. Airport schedules should be robust so they can handle delays and early arrivals. The impact of a flight that does not arrive or depart on schedule on other flights should be limited to the extent possible. This ensures that the knock-on effects of flight delays are reduced and better continuous operations can be achieved. Operating schedules should be planned in advance in case of flight delays. So this is a very important question not only for us who fly, but also for airport scheduling.

[ ]:

```

```

1.3 Computational Methods [10 pts]

Briefly describe the specific approach you will take to solve some concrete aspect of the general problem.

You should include all the numerical or computational methods you intend to use. These can include methods or packages we did not discuss in class but provide some reference to the method. You do not need to explain how the methods work, but you should briefly justify your choices.

If you need to install or import any additional python packages, please provide complete installation instructions in the code block below

Since the main field of this project is machine learning, we will mainly use the sklearn library and some functions.

1. import math for doing some basic manipulation.
2. import numpy for doing some testing.
3. RandomForestClassifier for fitting the random forest model.

We're not going to go into the details of what a random forest is, we just need to know that a random forest is a model of many decision trees. Instead of simply averaging the prediction of a tree (such an algorithm can be called a "forest"), this model uses two key concepts that give it the name random forest:

i. Random sampling of training data points during tree construction
ii. Random feature subset considered in node segmentation

4. confusion_matrix for producing the confusion matrix.
5. precision_score and for producing the precision score:

Since it is a classification index, we can think of the accuracy rate, which is defined as the percentage of correct predicted results in the total sample, and the formula is as follows:

$$Accuracy = (TP + TN)/((TP, TN) + (FP + FN))$$

Although the accuracy rate can judge the total accuracy rate, it sometimes cannot be used as a good indicator to measure the results in the case of imbalanced samples. Let's say that in a total sample, 90 percent of the positive sample and 10 percent of the negative sample, the sample is severely unbalanced. In this case, we only need to predict all the samples as positive samples to get a high accuracy rate of 90%. 4. recall_score for producing the recall score. Here, recall, also known as the Recall rate, refers to the probability of being predicted as a positive sample in a sample that is actually positive, and its formula is as follows:

The recall rate = $\frac{TP}{(TP+FN)}$

6. roc_curve for producing the roc curve

Basic reference for ROC curve: Receiver Operating Characteristic (ROC) curve also known as Receiver Operating Characteristic curve. The curve was first used in radar signal detection to distinguish signal from noise. Later, it was used to evaluate the predictive power of the model, and the ROC curve was based on the obfuscation matrix. The two main indicators in the ROC curve are the true rate and the false positive rate, and the benefits of this choice are explained above. Where the abscissa is the false positive rate (FPR) and the ordinate is the true rate (TPR) Here, we define:

$$Sensitivity = TP/(TP + FN)$$

$$Specificity = TN/(FP + TN)$$

Then we have:

$$TPR = Sensitivity = TP/(TP + FN)$$

$$FPR = 1 - Specificity = FP/(FP + TN)$$

7. roc_auc_score for producing the auc score:

To calculate points on the ROC curve, we can evaluate the logistic regression model multiple times using different classification thresholds, but this is very inefficient. Fortunately, there is an efficient sorting algorithm called Area Under Curve that can provide us with such information. If we connect the diagonal, its area is exactly 0.5. The actual meaning of the diagonal line is: to judge the response and non-response randomly, the positive and negative sample coverage rate should be 50%, indicating random effect. The steeper the ROC curve, the better, so the ideal value is 1, a square, and the worst random judgment has 0.5, so the general AUC value is between 0.5 and 1.

8. precision_recall_curve for producing the precision recall curve:

The precision recall curve is similar to the ROC curve. The ROC curve is the line connecting the points of FPR and TPR, and the precision recall curve is the line connecting the points of accuracy and recall. We also know that Recall=TPR, so the horizontal ordinate of PRC is the vertical coordinates of ROC.

[2]:
```
# Provide complete installation or import information for external packages or␣
↪modules here e.g.

#pip install somepackage
# from somepackage import blah
import math
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import linear_model,metrics,svm
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix, precision_recall_fscore_support
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_curve

import sys
!{sys.executable} -m pip install yellowbrick
from yellowbrick.classifier import precision_recall_curve
```

```
Requirement already satisfied: yellowbrick in /opt/conda/lib/python3.8/site-
packages (1.3.post1)
Requirement already satisfied: scikit-learn>=0.20 in
/opt/conda/lib/python3.8/site-packages (from yellowbrick) (0.24.1)
Requirement already satisfied: numpy<1.20,>=1.16.0 in
/opt/conda/lib/python3.8/site-packages (from yellowbrick) (1.19.5)
Requirement already satisfied: cycler>=0.10.0 in /opt/conda/lib/python3.8/site-
packages (from yellowbrick) (0.10.0)
Requirement already satisfied: scipy>=1.0.0 in /opt/conda/lib/python3.8/site-
packages (from yellowbrick) (1.6.1)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in
/opt/conda/lib/python3.8/site-packages (from yellowbrick) (3.3.4)
Requirement already satisfied: six in /opt/conda/lib/python3.8/site-packages
(from cycler>=0.10.0->yellowbrick) (1.15.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.8/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.8/site-
packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (8.1.2)
Requirement already satisfied: python-dateutil>=2.1 in
/opt/conda/lib/python3.8/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/opt/conda/lib/python3.8/site-packages (from
matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.4.7)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.8/site-packages (from scikit-learn>=0.20->yellowbrick)
(2.1.0)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.8/site-
packages (from scikit-learn>=0.20->yellowbrick) (1.0.1)
```

1.4 Implementation [60 pts]

Use the Markdown and Code blocks below to implement and document your methods including figures. Only the first code block will be a grading cell but please add (not copy) cells in this section to organize your work.

Please make the description of your problem readable by interlacing clear explanatory text with code. All code should be well described and commented.

For at least one routine you code below, you should provide a test block (e.g. that implements `numpy.testing` routines) to validate your code.

Original data:

[3]:
```
# Get the original data set
#flights_data = pd.read_csv('pnwflights14.csv')
#flights_data.head()
flights_data = pd.read_csv('https://raw.githubusercontent.com/ismayc/
 ↪pnwflights14/master/data/flights.csv')
flights_data.head()
```

[3]:
```
   year  month  day  dep_time  dep_delay  arr_time  arr_delay carrier tailnum  \
0  2014      1    1       1.0       96.0     235.0       70.0      AS  N508AS
1  2014      1    1       4.0       -6.0     738.0      -23.0      US  N195UW
2  2014      1    1       8.0       13.0     548.0       -4.0      UA  N37422
3  2014      1    1      28.0       -2.0     800.0      -23.0      US  N547UW
4  2014      1    1      34.0       44.0     325.0       43.0      AS  N762AS

   flight origin dest  air_time  distance  hour  minute
0     145    PDX  ANC     194.0      1542   0.0     1.0
1    1830    SEA  CLT     252.0      2279   0.0     4.0
2    1609    PDX  IAH     201.0      1825   0.0     8.0
3     466    PDX  CLT     251.0      2282   0.0    28.0
4     121    SEA  ANC     201.0      1448   0.0    34.0
```

[4]:
```
flights_data.shape
```

[4]:
```
(162049, 16)
```

[5]:
```
flights_data.columns
```

[5]:
```
Index(['year', 'month', 'day', 'dep_time', 'dep_delay', 'arr_time',
       'arr_delay', 'carrier', 'tailnum', 'flight', 'origin', 'dest',
       'air_time', 'distance', 'hour', 'minute'],
      dtype='object')
```

[8]:
```
 #### In order to have a better understanding of our dataset, we will explain␣
 ↪the meaning of each column below.
 ## year stands for the year of departure
 ## month stands for the month of departure
 ## day stands for the day of departure
 ## dep_time stands for the schedualed departure local time
 ## dep_delay stands for the departure delay in minutes
 ## arr_time stands for the schedualed arrival local time
```

```
## arr_delay stands for the arrival delay in minutes
## carrier stands for the two letter carrier abbreviation
## tailnum stands for the plane tail number
## flight stands for the flight number
## origin stands for the origin
## dest stands for the destination
## air_time stands for the amount of time spent in the air
## distance stands for the distance between origin and destination
## hour stands for the time of scheduled departure broken into hour
## minute stands for the time of scheduled departure broken into minute
```

[6]:
```
flights_data.isnull().values.any()
```

[6]:
```
True
```

[7]:
```
flights_data.isnull().sum()
```

[7]:
```
year            0
month           0
day             0
dep_time      857
dep_delay     857
arr_time      988
arr_delay    1301
carrier         0
tailnum       248
flight          0
origin          0
dest            0
air_time     1301
distance        0
hour          857
minute        857
dtype: int64
```

[8]:
```
flights_data_witoutna = flights_data.dropna()
```

[9]:
```
flights_data_witoutna.isnull().values.any()
```

[9]:
```
False
```

[10]:
```
flights_data_witoutna.shape
```

[10]:
```
(160748, 16)
```

[11]:
```
# Do the data cleaning, delete non-sense rows of the entire data set.
# Remove the non-sense rows
```

```
flights_data_final = flights_data_witoutna.
 ↪drop(flights_data_witoutna[(flights_data_witoutna.dep_time <= 0)                                                 |␣
 ↪(flights_data_witoutna.air_time <= 0)                                                                           |␣
 ↪(flights_data_witoutna.distance <= 0)].index)
flights_data_final.shape
```

[11]:
```
(160748, 16)
```

[12]:
```
# Get the weather data set
weather_data = pd.read_csv('https://raw.githubusercontent.com/ismayc/
 ↪pnwflights14/master/data/weather.csv')
weather_data.head()
```

[12]:
```
  origin  year  month  day  hour   temp   dewp  humid  wind_dir  wind_speed  \
0    PDX  2014      1    1     0  44.96  41.00  85.90     290.0     3.45234
1    PDX  2014      1    1     1  44.96  39.92  82.38     290.0     6.90468
2    PDX  2014      1    1     2  44.06  39.92  85.25     330.0     5.75390
3    PDX  2014      1    1     3  44.06  39.92  85.25     280.0     6.90468
4    PDX  2014      1    1     4  42.98  41.00  92.65     290.0     5.75390

   wind_gust  precip  pressure  visib                 date
0   3.972884     0.0    1026.9    8.0  2014-01-01 00:00:00
1   7.945768     0.0    1027.3    9.0  2014-01-01 01:00:00
2   6.621473     0.0    1027.4    9.0  2014-01-01 02:00:00
3   7.945768     0.0    1027.6    9.0  2014-01-01 03:00:00
4   6.621473     0.0       NaN    7.0  2014-01-01 04:00:00
```

[13]:
```
merge_1 = pd.merge(flights_data_final, weather_data,␣
 ↪on=["origin","year","month","day","hour"])
merge_1 = merge_1.dropna()
merge_1.head()
```

[13]:
```
   year  month  day  dep_time  dep_delay  arr_time  arr_delay carrier tailnum  \
0  2014      1    1       1.0       96.0     235.0       70.0      AS  N508AS
1  2014      1    1       8.0       13.0     548.0       -4.0      UA  N37422
2  2014      1    1      28.0       -2.0     800.0      -23.0      US  N547UW
3  2014      1    1       4.0       -6.0     738.0      -23.0      US  N195UW
4  2014      1    1      34.0       44.0     325.0       43.0      AS  N762AS

   flight  …   temp   dewp  humid  wind_dir  wind_speed  wind_gust  precip  \
0     145  …  44.96  41.00  85.90     290.0     3.45234   3.972884     0.0
1    1609  …  44.96  41.00  85.90     290.0     3.45234   3.972884     0.0
2     466  …  44.96  41.00  85.90     290.0     3.45234   3.972884     0.0
3    1830  …  46.04  42.98  88.99       0.0     0.00000   0.000000     0.0
4     121  …  46.04  42.98  88.99       0.0     0.00000   0.000000     0.0
```

```
   pressure  visib                date
0    1026.9    8.0 2014-01-01 00:00:00
1    1026.9    8.0 2014-01-01 00:00:00
2    1026.9    8.0 2014-01-01 00:00:00
3    1028.2    6.0 2014-01-01 00:00:00
4    1028.2    6.0 2014-01-01 00:00:00

[5 rows x 26 columns]
```

[14]:

```python
# Merge these two data set. We can check the accuracy of our data set in this␣
↪way: we will reproduce the dep_time
# column and change the column value to hour by calculating which we will show␣
↪below. And we use numpy.testing to test
# if the column of dep_time and hour column are the same. If they are not the␣
↪same, an exception is raised at
# shape mismatch or conflicting values. In contrast to the standard usage in␣
↪numpy, NaNs are compared like numbers,
# no assertion is raised if both objects have NaNs in the same positions. But␣
↪here we don't need to consider this case
# since we have already omit the NA values.
# Next we will check the correctness of our dataset.
pd.options.mode.chained_assignment = None
for index, row in merge_1.iterrows():
    merge_1.loc[index, 'dep_time'] = math.floor(row['dep_time'] / 100)
merge_1.head()
```

[14]:

```
   year  month  day  dep_time  dep_delay  arr_time  arr_delay carrier tailnum  \
0  2014      1    1       0.0       96.0     235.0       70.0      AS  N508AS
1  2014      1    1       0.0       13.0     548.0       -4.0      UA  N37422
2  2014      1    1       0.0       -2.0     800.0      -23.0      US  N547UW
3  2014      1    1       0.0       -6.0     738.0      -23.0      US  N195UW
4  2014      1    1       0.0       44.0     325.0       43.0      AS  N762AS

   flight  …   temp   dewp  humid  wind_dir  wind_speed  wind_gust  precip  \
0     145  …  44.96  41.00  85.90     290.0     3.45234   3.972884     0.0
1    1609  …  44.96  41.00  85.90     290.0     3.45234   3.972884     0.0
2     466  …  44.96  41.00  85.90     290.0     3.45234   3.972884     0.0
3    1830  …  46.04  42.98  88.99       0.0     0.00000   0.000000     0.0
4     121  …  46.04  42.98  88.99       0.0     0.00000   0.000000     0.0

   pressure  visib                date
0    1026.9    8.0 2014-01-01 00:00:00
1    1026.9    8.0 2014-01-01 00:00:00
2    1026.9    8.0 2014-01-01 00:00:00
3    1028.2    6.0 2014-01-01 00:00:00
4    1028.2    6.0 2014-01-01 00:00:00
```

[5 rows x 26 columns]

[15]:
```
merge_1.shape
```

[15]: (130530, 26)

[16]:
```
# Here we are going to check the quality of dataset
hour = merge_1["hour"]
hour.shape
```

[16]: (130530,)

[17]:
```
dep_time_hour = merge_1["dep_time"]
dep_time_hour.shape
```

[17]: (130530,)

[18]:
```
# Here we will implement numpy.testing to check if the two columns are equal,␣
 ↪that is, to check the quality of our data
# set. Thus we found that our data set seems ok since our definition of column␣
 ↪of hour is the time of scheduled
# departure broken into hour. And here we produce the hour by ourselves and␣
 ↪found that they are equal.
# Thus we successfully check the correctness of our data set.
numpy.testing.assert_array_equal(hour, dep_time_hour)
# As we can see that there is no exception thrown, which indicates that the two␣
 ↪columns are the same. And it infer
# that we can use our data set since our data set is of high quality.
```

[19]:
```
# When we start to look at the significance of each column of the entire data,
# we should note that not all data will affect our prediction, so we should␣
 ↪first determine which
# data are relevant to our prediction model. So we thought about it and created␣
 ↪a new data frame number
# that we thought would affect our prediction model. Then drop the rows with NA␣
 ↪data again.

merge_2 =␣
 ↪merge_1[["arr_delay","dep_delay","dep_time","arr_time","temp","dewp","humid","wind_speed","
                                        "wind_gust","precip","pressure","visib",
                                       ␣
 ↪"carrier","air_time","distance","minute","flight",
                                        "origin","dest","year","month","day"]]
merge2_witoutna = merge_2.dropna()
merge2_witoutna.head()
```

[19]:
```
   arr_delay  dep_delay  dep_time  arr_time   temp   dewp  humid  wind_speed  \
0       70.0       96.0       0.0     235.0  44.96  41.00  85.90     3.45234
1       -4.0       13.0       0.0     548.0  44.96  41.00  85.90     3.45234
2      -23.0       -2.0       0.0     800.0  44.96  41.00  85.90     3.45234
3      -23.0       -6.0       0.0     738.0  46.04  42.98  88.99     0.00000
4       43.0       44.0       0.0     325.0  46.04  42.98  88.99     0.00000

   wind_dir  wind_gust  …  carrier  air_time  distance minute  flight  \
0     290.0   3.972884  …       AS     194.0      1542    1.0     145
1     290.0   3.972884  …       UA     201.0      1825    8.0    1609
2     290.0   3.972884  …       US     251.0      2282   28.0     466
3       0.0   0.000000  …       US     252.0      2279    4.0    1830
4       0.0   0.000000  …       AS     201.0      1448   34.0     121

   origin  dest  year month day
0     PDX   ANC  2014     1   1
1     PDX   IAH  2014     1   1
2     PDX   CLT  2014     1   1
3     SEA   CLT  2014     1   1
4     SEA   ANC  2014     1   1

[5 rows x 23 columns]
```

[20]:
```python
# The data is now clean, but there is still room for further processing.
# The dep_time and arr_time in the data set represents the planned departure␣
↪time, but the time is too detailed
# (over 500 different values), affecting the accuracy of our machine learning␣
↪model. We could divide every
# number in this column by 100 and round it down, so 1030 would be 10 and 1925␣
↪would be 19, and there would
# only be 24 discrete values in this column.
pd.options.mode.chained_assignment = None

for index, row in merge2_witoutna.iterrows():
    merge2_witoutna.loc[index, 'arr_time'] = math.floor(row['arr_time'] / 100)
merge2_witoutna.head()
```

[20]:
```
   arr_delay  dep_delay  dep_time  arr_time   temp   dewp  humid  wind_speed  \
0       70.0       96.0       0.0       2.0  44.96  41.00  85.90     3.45234
1       -4.0       13.0       0.0       5.0  44.96  41.00  85.90     3.45234
2      -23.0       -2.0       0.0       8.0  44.96  41.00  85.90     3.45234
3      -23.0       -6.0       0.0       7.0  46.04  42.98  88.99     0.00000
4       43.0       44.0       0.0       3.0  46.04  42.98  88.99     0.00000

   wind_dir  wind_gust  …  carrier  air_time  distance minute  flight  \
0     290.0   3.972884  …       AS     194.0      1542    1.0     145
1     290.0   3.972884  …       UA     201.0      1825    8.0    1609
```

```
2     290.0   3.972884  …        US     251.0       2282   28.0      466
3       0.0   0.000000  …        US     252.0       2279    4.0     1830
4       0.0   0.000000  …        AS     201.0       1448   34.0      121

   origin  dest  year month day
0     PDX   ANC  2014     1   1
1     PDX   IAH  2014     1   1
2     PDX   CLT  2014     1   1
3     SEA   CLT  2014     1   1
4     SEA   ANC  2014     1   1

[5 rows x 23 columns]
```

[21]:
```
merge2_witoutna.columns
```

[21]:
```
Index(['arr_delay', 'dep_delay', 'dep_time', 'arr_time', 'temp', 'dewp',
       'humid', 'wind_speed', 'wind_dir', 'wind_gust', 'precip', 'pressure',
       'visib', 'carrier', 'air_time', 'distance', 'minute', 'flight',
       'origin', 'dest', 'year', 'month', 'day'],
      dtype='object')
```

[22]:
```
def label_arr_delay (row):
    if row['arr_delay'] >= 0 :
        return 1
    if row['arr_delay'] < 0 :
        return 0
```

[23]:
```
#merge2_witoutna["arr_delay_whether"]=merge2_witoutna.apply (lambda row:␣
 ↪label_arr_delay(row), axis=1)
merge2_witoutna["arr_delay"]=merge2_witoutna.apply (lambda row:␣
 ↪label_arr_delay(row), axis=1)
merge2_witoutna.shape
```

[23]:
```
(130530, 23)
```

[24]:
```
def label_dep_delay (row):
    if row['dep_delay'] >= 0 :
        return 1
    if row['dep_delay'] < 0 :
        return 0
```

[25]:
```
#merge2_witoutna["dep_delay_whether"]=merge2_witoutna.apply (lambda row:␣
 ↪label_dep_delay(row), axis=1)
merge2_witoutna["dep_delay"]=merge2_witoutna.apply (lambda row:␣
 ↪label_dep_delay(row), axis=1)
```

```
merge2_witoutna.shape
```

[25]: (130530, 23)

[26]:
```
merge2_witoutna.head()
```

[26]:
```
   arr_delay  dep_delay  dep_time  arr_time   temp   dewp  humid  wind_speed  \
0          1          1       0.0       2.0  44.96  41.00  85.90     3.45234
1          0          1       0.0       5.0  44.96  41.00  85.90     3.45234
2          0          0       0.0       8.0  44.96  41.00  85.90     3.45234
3          0          0       0.0       7.0  46.04  42.98  88.99     0.00000
4          1          1       0.0       3.0  46.04  42.98  88.99     0.00000

   wind_dir  wind_gust  …  carrier  air_time  distance minute  flight  \
0     290.0   3.972884  …       AS     194.0      1542    1.0     145
1     290.0   3.972884  …       UA     201.0      1825    8.0    1609
2     290.0   3.972884  …       US     251.0      2282   28.0     466
3       0.0   0.000000  …       US     252.0      2279    4.0    1830
4       0.0   0.000000  …       AS     201.0      1448   34.0     121

   origin  dest  year month day
0     PDX   ANC  2014     1   1
1     PDX   IAH  2014     1   1
2     PDX   CLT  2014     1   1
3     SEA   CLT  2014     1   1
4     SEA   ANC  2014     1   1

[5 rows x 23 columns]
```

[27]:
```
# Now we generate the origin, dest, carrier columns into the indicator column
# (split for each station) and delete the origin, dest, carrier columns␣
↪themselves
df_final = pd.get_dummies(merge2_witoutna, columns=['origin', 'dest','carrier'])
df_final.head()
```

[27]:
```
   arr_delay  dep_delay  dep_time  arr_time   temp   dewp  humid  wind_speed  \
0          1          1       0.0       2.0  44.96  41.00  85.90     3.45234
1          0          1       0.0       5.0  44.96  41.00  85.90     3.45234
2          0          0       0.0       8.0  44.96  41.00  85.90     3.45234
3          0          0       0.0       7.0  46.04  42.98  88.99     0.00000
4          1          1       0.0       3.0  46.04  42.98  88.99     0.00000

   wind_dir  wind_gust  …  carrier_AS  carrier_B6  carrier_DL  carrier_F9  \
0     290.0   3.972884  …           1           0           0           0
1     290.0   3.972884  …           0           0           0           0
2     290.0   3.972884  …           0           0           0           0
3       0.0   0.000000  …           0           0           0           0
```

```
4       0.0   0.000000  ...          1           0           0           0

   carrier_HA  carrier_OO  carrier_UA  carrier_US  carrier_VX  carrier_WN
0           0           0           0           0           0           0
1           0           0           1           0           0           0
2           0           0           0           1           0           0
3           0           0           0           1           0           0
4           0           0           0           0           0           0

[5 rows x 104 columns]
```

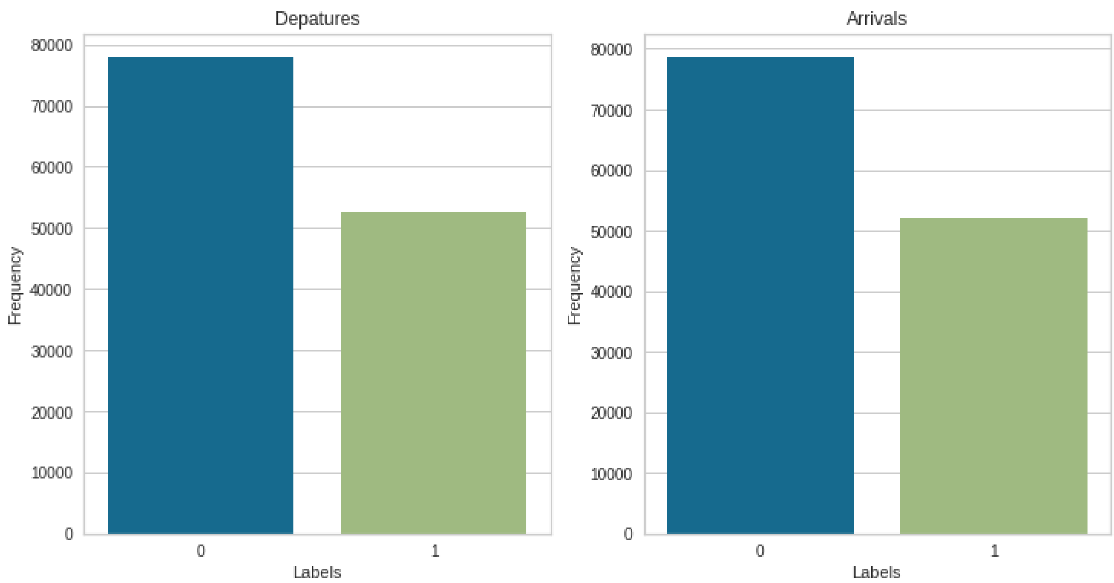
[28]:
```python
f, (ax,ax1) = plt.subplots(1,2, figsize=(12,6))
dep = sns.countplot(df_final['dep_delay'], ax=ax)
dep.set_title('Depatures')
dep.set_xlabel('Labels')
dep.set_ylabel('Frequency')

arr = sns.countplot(df_final['arr_delay'], ax=ax1)
arr.set_title('Arrivals')
arr.set_xlabel('Labels')
arr.set_ylabel('Frequency')

# From the graphs below, we can see a greater concentration of flights with␣
 ↪timely departures and arrivals. Another
# insight that we can observe is that the proportions are very similar in the␣
 ↪two variables, it is very likely that
# the departures or not in delay are very important for predictive modeling␣
 ↪about delayed arrivals. This aspect should
# also be taken into account in our final analysis
```

```
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

[28]:
```
Text(0, 0.5, 'Frequency')
```



[29]:
```
# To create a machine learning model, we need two data sets, one for training␣
↪and one for testing.
# Usually we only have one dataset, as in this case, so we split it into two.␣
↪Here, we can split the
# DataFrame 80-20. At the same time, we need to annotate the characteristic␣
↪column and the label column.
# The characteristic column is the column used as the input of the model (such␣
↪as the origin and destination),
# and the label column is the column we predicted, in this case␣
↪arr_delay_whether.
train_x, test_x, train_y, test_y = train_test_split(df_final.drop('arr_delay',␣
↪axis=1),
                                                    df_final['arr_delay'],␣
↪test_size=0.2, random_state=42)
test_x.shape
```

[29]: (26106, 103)

[30]:
```
#model_lr = linear_model.
↪LogisticRegression(penalty="l2",class_weight='auto',max_iter=1000000000000000)
#model_lr.fit(train_x,train_y)
model_rf = RandomForestClassifier(random_state=13)
model_rf.fit(train_x, train_y)
```

[30]: RandomForestClassifier(random_state=13)

[31]:
```
# We then used predict method to test the model against the values in test_x␣
↪and score method to
# determine the average accuracy of the model
#pr_lr = model_lr.predict(test_x)
predicted_rf = model_rf.predict(test_x)
model_rf.score(test_x, test_y)
```

[31]: 0.8005056308894507

[32]:
```
probabilities_rf = model_rf.predict_proba(test_x)
# General criteria for AUC
# 0.5-0.7: Less effective, but good enough for predicting stocks
# 0.7-0.85: general effect
# 0.85-0.95: Good effect
# 0.95-1: Very good, but generally unlikely
numpy.testing.assert_array_less([0.5,0.7,0.85,0.95], roc_auc_score(test_y,␣
↪probabilities_rf[:, 1]))
# By testing, we found that there is one mismatched element in the four numbers␣
↪and the only possibility
# for the mismatch element is 0.95. This is a good sign meaning that our auc␣
↪socre is in the range of 0.85-0.95
# which indicate that our model has good effect. Also We find that the AUC␣
↪score is higher than the average accuracy
# calculated above, because the output of score method reflects how many items␣
↪in the test set can be correctly
# predicted by the model. This score is influenced by the fact that the data␣
↪set used to finalize and test the
# model contains more rows representing on-time arrivals than late arrivals.
```

```
       ␣
↪---------------------------------------------------------------------------

       AssertionError                            Traceback (most recent call␣
↪last)

       <ipython-input-32-c5b6f96fd062> in <module>
         5 # 0.85-0.95: Good effect
         6 # 0.95-1: Very good, but generally unlikely
   ----> 7 numpy.testing.assert_array_less([0.5,0.7,0.85,0.95],␣
↪roc_auc_score(test_y, probabilities_rf[:, 1]))
         8 # By testing, we found that there is one mismatched element in the␣
↪four numbers and the only possibility
         9 # for the mismatch element is 0.95. This is a good sign meaning that␣
↪our auc socre is in the range of 0.85-0.95
```

```
        [… skipping hidden 1 frame]


    /opt/conda/lib/python3.8/site-packages/numpy/testing/_private/utils.py␣
↪in assert_array_compare(comparison, x, y, err_msg, verbose, header, precision,␣
↪equal_nan, equal_inf)
    838                                 verbose=verbose, header=header,
    839                                 names=('x', 'y'),␣
↪precision=precision)
--> 840             raise AssertionError(msg)
    841     except ValueError:
    842         import traceback


    AssertionError:
Arrays are not less-ordered

Mismatched elements: 1 / 4 (25%)
Max absolute difference: 0.3766347340087893
Max relative difference: 0.4296370191567302
 x: array([0.5 , 0.7 , 0.85, 0.95])
 y: array(0.876635)
```

[33]:
```
# We can learn more about the model's behavior by generating confusion matrices
# (also known as error matrices). The confusion matrix quantifies the number of␣
↪times each answer is correctly
# or incorrectly classified. Specifically, it quantifies the number of false␣
↪positives, false negatives,
# true and true negatives.
confusion_matrix(test_y, predicted_rf)
# The first line in the output represents on-time flights. The first column of␣
↪the line shows the number of flights
## that were correctly predicted to arrive on time, while the second column␣
↪shows the number of flights that were
# predicted to be late but were not. As a result, the model seems to be good at␣
↪predicting the arrival of flights
# on time. Then let's move on to the second line, which represents the delayed␣
↪flight. The first column shows the
# number of late flights that were wrongly predicted to arrive on time. The␣
↪second column shows the number of flights
# that were correctly predicted to be late. We found that our model was very␣
↪good at predicting that flights would
# arrive on time and predicting that flights would be late.
```

[33]:
```
array([[13691,  1961],
       [ 3247,  7207]])
```

[34]:

```python
# Here we would like to include accuracy and recall. Suppose the model
# is given three on-time arrivals and three late arrivals, and it correctly␣
↪predicts two of the on-time arrivals,
# but incorrectly predicts two of the late arrivals. In this case, the accuracy␣
↪rate would be 50%
# (two of the four flights it classified as on-time arrivals actually arrived␣
↪on time), while its recall
# rate would be 67% (it correctly identified two of the three on-time arrivals).
train_predictions_rf = model_rf.predict(train_x)
precision_score(train_y, train_predictions_rf)
```

[34]: 1.0

[35]:

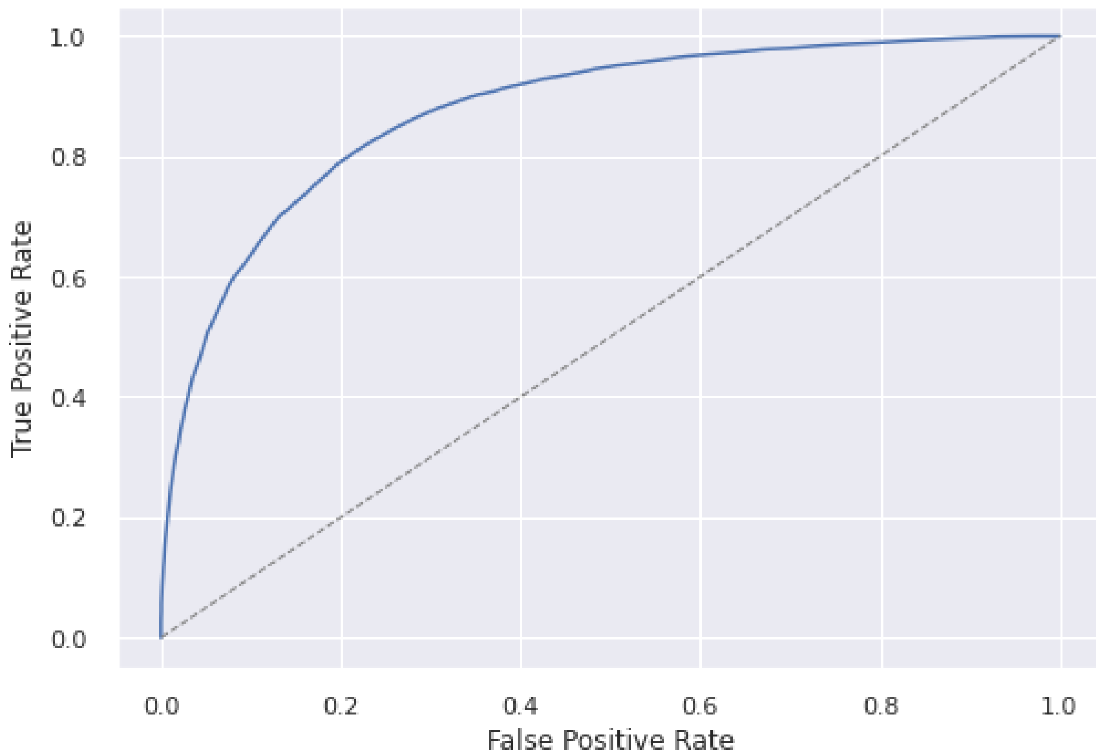
```python
recall_score(train_y, train_predictions_rf)
```

[35]: 1.0

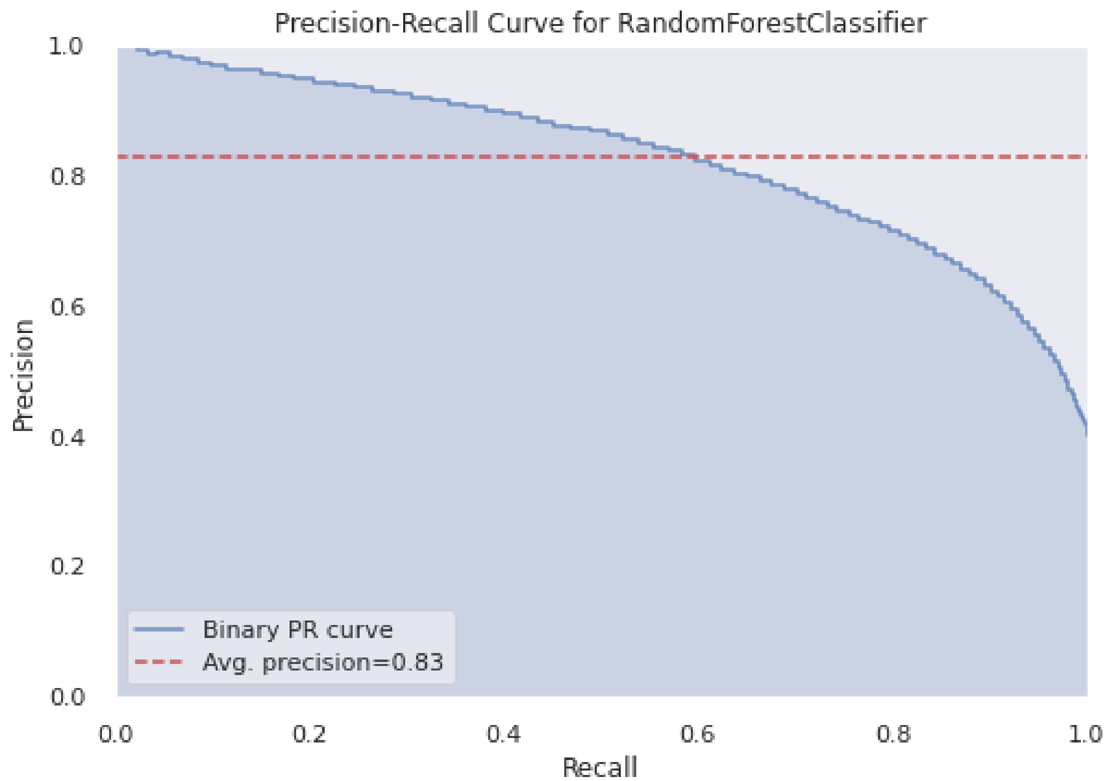
[36]:

```python
# ROC curve
sns.set()
fpr, tpr, _ = roc_curve(test_y, probabilities_rf[:, 1])
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], color='grey', lw=1, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# The dotted line in the middle of the graph indicates a 50% chance of getting␣
↪the right answer.
# The blue curve shows the accuracy of the model.
```

[36]: Text(0, 0.5, 'True Positive Rate')



[37]:
```
# PR curve
pr_curve = precision_recall_curve(model_rf, train_x,
                                  train_y, test_x, test_y)
# The diagram below shows us the tradeoff between accuracy and recall. If we␣
↪seek a larger recall in
# favor of our positive classes, we will sacrifice the precision of the model␣
↪which make sense.
```



1.5 Discussion [10 pts]

Discuss the results of your code including * Why do you believe that your numerical results are correct (convergence, test cases etc)? * Did the project work (in your opinion)? * If yes: what would be the next steps to try * If no: Explain why your approach did not work and what you would do differently next time

We believe that our numerical results are correct. We first read the PNWFlights14 data set, and then manually added a weather data set, so we used a lot of variables for prediction and increased the accuracy of prediction. Then, depending on the definition of the columns in the dataset, we can use one of the column to calculate the other column to check whether the dataset is a mismatch or not. Numpy.test didn't give us feedback(throw an exception), which means that our dataset has a high quality. We also cleared the data of NA in the data set and manually processed non-sense data points, which also contributed to the accuracy of our prediction. Finally, our model was tested by numpy.test, and our prediction accuracy was within the range of 0.85 and 0.95, with excellent performance. By looking at the ROC curve, our model performed really well. Also we plot the PR curve and shows us the tradeoff between accuracy and recall. If we seek a larger recall in favor of our positive classes, we will sacrifice the precision of the model which make sense. Overall, we think our project work and actually can make good predictions. Our next step is to try to use cross-validation to select variables and see if it makes the prediction more accurate. Of course, cross-validation may have a very long run cost, but the result may be a simplified model and higher accuracy. We should also check whether our model has overfit. Of course, we can also choose other machine learning methods such as Nueral Network to build models and compare accuracy.

```
[ ]:
```