*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

Assignment 4 - Understanding and Predicting Property Maintenance Fines

This assignment is based on a data challenge from the Michigan Data Science Team (MDST).

The Michigan Data Science Team (MDST) and the Michigan Student Symposium for Interdisciplinary Statistical Sciences (MSSISS) have partnered with the City of Detroit to help solve one of the most pressing problems facing Detroit - blight. Blight violations are issued by the city to individuals who allow their properties to remain in a deteriorated condition. Every year, the city of Detroit issues millions of dollars in fines to residents and every year, many of these fines remain unpaid. Enforcing unpaid blight fines is a costly and tedious process, so the city wants to know: how can we increase blight ticket compliance?

The first step in answering this question is understanding when and why a resident might fail to comply with a blight ticket. This is where predictive modeling comes in. For this assignment, your task is to predict whether a given blight ticket will be paid on time.

All data for this assignment has been provided to us through the Detroit Open Data Portal. **Only the data already included in your Coursera directory can be used for training the model for this assignment.** Nonetheless, we encourage you to look into data from other Detroit datasets to help inform feature creation and model selection. We recommend taking a look at the following related datasets:

- Building Permits
- Trades Permits
- Improve Detroit: Submitted Issues
- DPD: Citizen Complaints
- Parcel Map

We provide you with two data files for use in training and validating your models: train.csv and test.csv. Each row in these two files corresponds to a single blight ticket, and includes information about when, why, and to whom each ticket was issued. The target variable is compliance, which is True if the ticket was paid early, on time, or within one month of the hearing data, False if the ticket was paid after the hearing date or not at all, and Null if the violator was found not responsible. Compliance, as well as a handful of other variables that will not be available at test-time, are only included in train.csv.

Note: All tickets where the violators were found not responsible are not considered during evaluation. They are included in the training set as an additional source of data for visualization, and to enable unsupervised and semi-supervised approaches. However, they are not included in the test set.

File descriptions (Use only this data for training your model!)

```
readonly/train.csv - the training set (all tickets issued 2004-2011)
readonly/test.csv - the test set (all tickets issued 2012-2016)
readonly/addresses.csv & readonly/latlons.csv - mapping from ticket id to addresses, and from addresses to lat/lon coordi
nates.
 Note: misspelled addresses may be incorrectly geolocated.
```

Data fields

train.csv & test.csv

```
ticket_id - unique identifier for tickets
agency_name - Agency that issued the ticket
inspector_name - Name of inspector that issued the ticket
violator_name - Name of the person/organization that the ticket was issued to
violation_street_number, violation_street_name, violation_zip_code - Address where the violation occurred
mailing_address_str_number, mailing_address_str_name, city, state, zip_code, non_us_str_code, country - Mailing address o
f the violator
ticket_issued_date - Date and time the ticket was issued
hearing_date - Date and time the violator's hearing was scheduled
violation_code, violation_description - Type of violation
disposition - Judgment and judgement type
fine_amount - Violation fine amount, excluding fees
admin_fee - $20 fee assigned to responsible judgments
state_fee - $10 fee assigned to responsible judgments
late_fee - 10% fee assigned to responsible judgments
discount_amount - discount applied, if any
clean_up_cost - DPW clean-up or graffiti removal cost
judgment_amount - Sum of all fines and fees
grafitti_status - Flag for graffiti violations
```

train.csv only

```
payment_amount - Amount paid, if any
payment_date - Date payment was made, if it was received
payment_status - Current payment status as of Feb 1 2017
balance_due - Fines and fees still owed
collection_status - Flag for payments in collections
compliance [target variable for prediction]
 Null = Not responsible
 0 = Responsible, non-compliant
 1 = Responsible, compliant
compliance_detail - More information on why each ticket was marked compliant or non-compliant
```

Evaluation

Your predictions will be given as the probability that the corresponding blight ticket will be paid on time.

The evaluation metric for this assignment is the Area Under the ROC Curve (AUC).

Your grade will be based on the AUC score computed for your classifier. A model which with an AUROC of 0.7 passes this assignment, over 0.75 will recieve full points.

For this assignment, create a function that trains a model to predict blight ticket compliance in Detroit using `readonly/train.csv`. Using this model, return a series of length 61001 with the data being the probability that each corresponding ticket from `readonly/test.csv` will be paid, and the index being the ticket_id.

Example:

```
ticket_id
   284932    0.531842
   285362    0.401958
   285361    0.105928
   285338    0.018572
             ...
   376499    0.208567
   376500    0.818759
   369851    0.018528
   Name: compliance, dtype: float32
```

Hints

- Make sure your code is working before submitting it to the autograder.
- Print out your result to see whether there is anything weird (e.g., all probabilities are the same).
- Generally the total runtime should be less than 10 mins. You should NOT use Neural Network related classifiers (e.g., MLPClassifier) in this question.
- Try to avoid global variables. If you have other functions besides blight_model, you should move those functions inside the scope of blight_model.
- Refer to the pinned threads in Week 4's discussion forum when there is something you could not figure it out.

In [15]:
```
#%matplotlib notebook
import pandas as pd
import numpy as np
#import matplotlib.pyplot as plt
pd.set_option('display.max_columns',50)

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score

def blight_model():
    # Reading files:
    train = pd.read_csv('train.csv',encoding='ISO-8859-1')
    test = pd.read_csv('test.csv',encoding='ISO-8859-1')
    test.set_index(test['ticket_id'],inplace=True)

    # Cleaning:
    train.dropna(subset=['compliance'],inplace=True)
    train = train[train['country']=='USA']
    #test = test[test['country']=='USA']

    label_encoder = LabelEncoder()
    label_encoder.fit(train['disposition'].append(test['disposition'], ignore_index=True))
    train['disposition'] = label_encoder.transform(train['disposition'])
    test['disposition'] = label_encoder.transform(test['disposition'])

    label_encoder = LabelEncoder()
    label_encoder.fit(train['violation_code'].append(test['violation_code'], ignore_index=True))
    train['violation_code'] = label_encoder.transform(train['violation_code'])
    test['violation_code'] = label_encoder.transform(test['violation_code'])

    feature_names=['disposition','violation_code']
    X = train[feature_names]
    y = train['compliance']
    test = test[feature_names]
    X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)

    # grid search
    model = RandomForestRegressor()
    param_grid = {'n_estimators':[5,7], 'max_depth':[5,10]}
    grid_search = GridSearchCV(model, param_grid, scoring="roc_auc")
    grid_result = grid_search.fit(X_train, y_train)

    # summarize results
    print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
    return pd.DataFrame(grid_result.predict(test),index=test.index,columns=['compliance'])
```

In [16]:
```
blight_model()
```

```
/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2827: DtypeWarning: Columns (11,12,31) have mixed type
s. Specify dtype option on import or set low_memory=False.
  if self.run_code(code, result):
```

```
Best: 0.772353 using {'max_depth': 10, 'n_estimators': 7}
```

Out[16]:

	compliance
ticket_id	
284932	0.160314
285362	0.028671
285361	0.060922
285338	0.028671
285346	0.082971
285345	0.028671
285347	0.067833
285342	0.475521
285530	0.028671
284989	0.028671
285344	0.050240
285343	0.028671
285340	0.028671
285341	0.050240
285349	0.082971
285348	0.028671

284991	0.028671
285532	0.028671
285406	0.028671
285001	0.038514
285006	0.044235
285405	0.028671
285337	0.028671
285496	0.050240
285497	0.028671
285378	0.028671
285589	0.028671
285585	0.028671
285501	0.060922
285581	0.028671
...	...
376367	0.044235
376366	0.038514
376362	0.263067
376363	0.393333
376365	0.044235
376364	0.038514
376228	0.038514
376265	0.038514
376286	0.326283
376320	0.038514
376314	0.038514
376327	0.326283
376385	0.326283
376435	0.005102
376370	0.263067
376434	0.067833
376459	0.244100
376478	0.031423
376473	0.038514
376484	0.012511
376482	0.014592
376480	0.012511
376479	0.012511
376481	0.012511
376483	0.019449
376496	0.009287
376497	0.009287
376499	0.082971
376500	0.082971
369851	0.050240

61001 rows × 1 columns

In [ ]: