Assignment 4

April 26, 2021

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.*

0.1 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
 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
mailing_address_str_number, mailing_address_str_name, city, state, zip_code, non_us
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
```

0.2 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
```

0.2.1 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 [3]:
```python
import pandas as pd
import numpy as np



df_train = pd.read_csv('readonly/train.csv', encoding = "ISO-8859-1")
df_test = pd.read_csv('readonly/test.csv', encoding = "ISO-8859-1")
```

```
columns_to_remove_train = ['balance_due',
 'collection_status',
 'compliance_detail',
 'payment_amount',
 'payment_date',
 'payment_status']

columns_to_remove_all = ['violator_name', 'zip_code', 'country', 'city',
                     'inspector_name', 'violation_street_number', 'violati
                     'violation_zip_code', 'violation_description',
                     'mailing_address_str_number', 'mailing_address_str_na
                     'non_us_str_code',
                     'ticket_issued_date', 'hearing_date','grafitti_status


df_train.drop(columns_to_remove_train, axis=1, inplace=True)
df_train.drop(columns_to_remove_all, axis=1, inplace=True)
df_test.drop(columns_to_remove_all, axis=1, inplace=True)


df_latlons = pd.read_csv('readonly/latlons.csv')

df_address =  pd.read_csv('readonly/addresses.csv')

df_id_latlons = pd.merge(df_latlons,df_address,on='address')


df_train = pd.merge(df_train,df_id_latlons,on='ticket_id')
df_test = pd.merge(df_test,df_id_latlons,on='ticket_id')
vio_code_freq_top15 = df_train['violation_code'].value_counts().index[0:15]
#print(vio_code_freq_top15)

df_train['violation_code_freq_top15'] = [list(vio_code_freq_top15).index(c)
#print(df_train)
# drop violation code

df_train.drop('violation_code', axis=1, inplace=True)

df_test['violation_code_freq_top15'] = [list(vio_code_freq_top15).index(c)
df_test.drop('violation_code', axis=1, inplace=True)


df_train = df_train[df_train.compliance.isnull() == False]

df_train.lat.fillna(method='pad', inplace=True)
df_train.lon.fillna(method='pad', inplace=True)
df_train.state.fillna(method='pad', inplace=True)
#print(df_train)
```

```
df_test.lat.fillna(method='pad', inplace=True)
df_test.lon.fillna(method='pad', inplace=True)
df_test.state.fillna(method='pad', inplace=True)

df_train.drop('address', axis=1, inplace=True)
df_test.drop('address', axis=1, inplace=True)
```

```
#one_hot_encode_columns = ['agency_name', 'state', 'disposition']
#[ df_train[c].unique().size for c in one_hot_encode_columns]

#df_train = pd.get_dummies(df_train, columns=one_hot_encode_columns)
#df_test = pd.get_dummies(df_test, columns=one_hot_encode_columns)
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
# creating instance of labelencoder
labelencoder = LabelEncoder()
# Assigning numerical values and storing in another column
#df_train = labelencoder.fit_transform(df_train(one_hot_encode_columns, axi
#df_test = labelencoder.fit_transform(df_test(one_hot_encode_columns, axis=
#enc = OneHotEncoder(handle_unknown='ignore')

df_train.set_index('ticket_id', inplace=True)
df_test.set_index('ticket_id', inplace=True)
for col in df_train.columns[df_train.dtypes == "object"]:
    df_train[col] = labelencoder.fit_transform(df_train[col])
for col in df_test.columns[df_test.dtypes == "object"]:
    df_test[col] = labelencoder.fit_transform(df_test[col])
#print(df_test)
#X_test = df_test
X = df_train.drop('compliance',axis=1)
columns_to_use = list(X.columns.values)
df_test = df_test[columns_to_use]
X_test = df_test
y = df_train['compliance']
```

```
#from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
X_train, X_test_2, y_train, y_test = train_test_split(X, y, random_state=0,
#print(X_train.shape, X_test.shape)
```

```python
def blight_model():

    # Your code here
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.ensemble import GradientBoostingClassifier
    from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import train_test_split
    clf_RF = RandomForestClassifier(max_features = 9, random_state = 0)
    clf_RF.fit(X_train, y_train)
    clf_GDBT= GradientBoostingClassifier(learning_rate = 0.1, max_depth = 1
    clf_GDBT.fit(X_train, y_train)
    print('Here training score RF: {}'.format(clf_RF.score(X_train, y_train
    print('Here test score RF: {}'.format(clf_RF.score(X_test_2, y_test)))
    print('Here training score GDBT: {}'.format(clf_GDBT.score(X_train, y_t
    print('Here test score GDBT: {}'.format(clf_GDBT.score(X_test_2, y_test
    roc_score_GDBT = roc_auc_score(y_test, clf_GDBT.predict_proba(X_test_2)
    roc_score_RF = roc_auc_score(y_test, clf_RF.predict_proba(X_test_2)[:,1
    print('roc_score_GDBT : {}'.format(roc_score_GDBT))
    print('roc_score_RF : {}'.format(roc_score_RF))
    result = pd.Series(data=clf_GDBT.predict_proba(X_test)[:,1], index=X_te
    #print(len(preds))


    #return roc_score_GDBT, roc_score_RF#preds# Your answer here
    return result
print(blight_model())
```

```
/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2717: Dtype
  interactivity=interactivity, compiler=compiler, result=result)


Here training score RF: 0.9839155682551387
Here test score RF: 0.9373279959969978
Here training score GDBT: 0.9485996076004372
Here test score GDBT: 0.9435826870152615
roc_score_GDBT : 0.8247317767340392
roc_score_RF : 0.7567272036634716
ticket_id
284932    0.710080
285362    0.063799
285361    0.210964
285338    0.156551
285346    0.105583
285345    0.105583
```

285347	0.105526
285342	0.905433
285530	0.030195
284989	0.235299
285344	0.189859
285343	0.219612
285340	0.364025
285341	0.106245
285349	0.076645
285348	0.076645
284991	0.157556
285532	0.063814
285406	0.113215
285001	0.063057
285006	0.027711
285405	0.076214
285337	0.048713
285496	0.178914
285497	0.178914
285378	0.079893
285589	0.018918
285585	0.165298
285501	0.125347
285581	0.061870
	...
376367	0.044942
376366	0.198425
376362	0.146759
376363	0.146759
376365	0.044942
376364	0.198425
376228	0.236039
376265	0.192293
376286	0.902197
376320	0.163682
376314	0.141464
376327	0.923565
376385	0.933744
376435	0.930786
376370	0.923565
376434	0.120893
376459	0.104999
376478	0.029294
376473	0.138600
376484	0.222541
376482	0.043401
376480	0.047525
376479	0.047525

```
376481    0.047525
376483    0.137490
376496    0.035498
376497    0.035498
376499    0.212780
376500    0.216210
369851    0.925106
dtype: float64
```

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

```
In [ ]:
```