Assignment 4

May 23, 2019

---

*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]:
```
import pandas as pd
import numpy as np

def blight_model():

    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import roc_auc_score
    from sklearn.ensemble import GradientBoostingClassifier
    from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.model_selection import GridSearchCV

#loading the data
#print('loading the data')
train = pd.read_csv('train.csv', encoding = "ISO-8859-1")
test = pd.read_csv('test.csv', encoding = "ISO-8859-1")
addresses = pd.read_csv('addresses.csv', encoding = "ISO-8859-1")

#Merge the addresses and lat/lons into the train and test DataFrames
train = pd.merge(train, addresses, how='inner', left_on='ticket_id', ri
test = pd.merge(test, addresses, how='inner', left_on='ticket_id', righ

#Remove records from train data with 'compliance' == NaN
train = train.dropna(subset=['compliance'])
train['compliance'] = train['compliance'].astype(int)


#Convert NaN to "NA" in columns to convert to type category
convert_columns={'country': 'category',
                 'non_us_str_code': 'category',
                 'compliance': 'category',
                 'state': 'category',
                 'zip_code': 'category'
                }

for df in [test, train]:
    for col, col_type in convert_columns.items():
        if col in df:
            if col_type == 'category':
                df[col] = df[col].replace(np.nan, "NA", regex=True).ast
            elif col_type == 'int':
                df[col] = df[col].replace(np.nan, 0, regex=True).astype


#Remove unneeded columns from X
#print('dropping unnecessaary columns')
common_cols_to_drop = ['agency_name', 'inspector_name', 'mailing_addres
                       'violator_name', 'violation_street_number', 'vio
                       'mailing_address_str_name', 'address', 'admin_fe
                       'state_fee', 'late_fee', 'ticket_issued_date', '
                       'fine_amount', 'clean_up_cost', 'disposition', '
                       'violation_code', 'city']
train_cols_to_drop = ['payment_status', 'payment_date', 'balance_due',
train = train.drop(train_cols_to_drop, axis=1).set_index('ticket_id')
test = test.drop(common_cols_to_drop, axis=1).set_index('ticket_id')
y_train = train['compliance']
X_train_cols_to_drop = ['compliance', 'compliance_detail', 'collection_
train = train.drop(X_train_cols_to_drop, axis=1)
```

```
        cat_columns = train.select_dtypes(['category']).columns
        for df in [test, train]:
            df[cat_columns] = df[cat_columns].apply(lambda x: x.cat.codes)

        #creating xtrain data

        X_train = train.copy()

        #print('classifier')

        grid_values = {'learning_rate': [0.01, 0.1, 1], 'max_depth': [3, 4, 5]}
        clf = GradientBoostingClassifier(random_state = 0)
        grid_clf_auc = GridSearchCV(clf, param_grid = grid_values, scoring = 'r

        #print('training')

        grid_clf_auc.fit(X_train, y_train)

        #print('predicting')

        probs = grid_clf_auc.predict_proba(test)[:, 1]

        #print('returning')

        result = pd.Series(probs, index=test.index)

        return result
```

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

```
loading the data


/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2827: Dtype
  if self.run_code(code, result):


dropping unnecessaary columns
classifier
training
predicting
returning
```

Out[ ]:
```
ticket_id
284932    0.205176
285362    0.094864
285361    0.228263
```

285338	0.205176
285346	0.228263
285345	0.308038
285347	0.173571
285342	0.608790
285530	0.279054
284989	0.048395
285344	0.176957
285343	0.094864
285340	0.094864
285341	0.176957
285349	0.181394
285348	0.145468
284991	0.106330
285532	0.119426
285406	0.119426
285001	0.119426
285006	0.094864
285405	0.065633
285337	0.439720
285496	0.176957
285497	0.205176
285378	0.039963
285589	0.107934
285585	0.205176
285501	0.228263
285581	0.092503
	...
376367	0.119426
376366	0.112573
376362	0.112573
376363	0.205176
376365	0.119426
376364	0.112573
376228	0.112573
376265	0.112573
376286	0.553783
376320	0.112573
376314	0.112573
376327	0.553783
376385	0.553783
376435	0.119426
376370	0.553783
376434	0.176957
376459	0.228263
376478	0.042838
376473	0.107662
376484	0.084503

```
376482    0.109218
376480    0.109218
376479    0.109218
376481    0.109218
376483    0.102882
376496    0.094864
376497    0.094864
376499    0.228263
376500    0.228263
369851    0.486462
dtype: float64
```

```
In [ ]:
```