*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 (https://www.coursera.org/learn/python-machine-learning/resources/bANLa)](https://www.coursera.org/learn/python-machine-learning/resources/bANLa) 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 (http://midas.umich.edu/mdst/)](http://midas.umich.edu/mdst/)).

The Michigan Data Science Team ([MDST (http://midas.umich.edu/mdst/)](http://midas.umich.edu/mdst/)) and the Michigan Student Symposium for Interdisciplinary Statistical Sciences ([MSSISS (https://sites.lsa.umich.edu/mssiss/)](https://sites.lsa.umich.edu/mssiss/)) have partnered with the City of Detroit to help solve one of the most pressing problems facing Detroit - blight. [Blight violations (http://www.detroitmi.gov/How-Do-I/Report/Blight-Complaint-FAQs)](http://www.detroitmi.gov/How-Do-I/Report/Blight-Complaint-FAQs) 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 (https://data.detroitmi.gov/)](https://data.detroitmi.gov/). **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 (https://data.detroitmi.gov/Property-Parcels/Building-Permits/xw2a-a7tf)](https://data.detroitmi.gov/Property-Parcels/Building-Permits/xw2a-a7tf)
- [Trades Permits (https://data.detroitmi.gov/Property-Parcels/Trades-Permits/635b-dsgv)](https://data.detroitmi.gov/Property-Parcels/Trades-Permits/635b-dsgv)
- [Improve Detroit: Submitted Issues (https://data.detroitmi.gov/Government/Improve-Detroit-Submitted-Issues/fwz3-w3yn)](https://data.detroitmi.gov/Government/Improve-Detroit-Submitted-Issues/fwz3-w3yn)
- [DPD: Citizen Complaints (https://data.detroitmi.gov/Public-Safety/DPD-Citizen-Complaints-2016/kahe-efs3)](https://data.detroitmi.gov/Public-Safety/DPD-Citizen-Complaints-2016/kahe-efs3)
- [Parcel Map (https://data.detroitmi.gov/Property-Parcels/Parcel-Map/fxkw-udwf)](https://data.detroitmi.gov/Property-Parcels/Parcel-Map/fxkw-udwf)

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!)

```
    train.csv - the training set (all tickets issued 2004-2011)
    test.csv - the test set (all tickets issued 2012-2016)
    addresses.csv & latlons.csv - mapping from ticket id to addresses, and from addresses to lat/lon coordinates.
     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 of 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 `train.csv`. Using this model, return a series of length 61001 with the data being the probability that each corresponding ticket from `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 [ ]:  import pandas as pd
         import numpy as np
         def blight_model():
             from datetime import datetime
             import pandas as pd
             import numpy as np
             from sklearn.ensemble import RandomForestClassifier
             from sklearn.preprocessing import MinMaxScaler
             from sklearn.model_selection import train_test_split, GridSearchCV
```

```python
    from sklearn.metrics import roc_auc_score

    def time_gap(hearing_date_str, ticket_issued_date_str):
        if not hearing_date_str or type(hearing_date_str)!=str: return 73
        hearing_date = datetime.strptime(hearing_date_str, "%Y-%m-%d %H:%M:%S")
        ticket_issued_date = datetime.strptime(ticket_issued_date_str, "%Y-%m-%d %H:%M:%S")
        gap = hearing_date - ticket_issued_date
        return gap.days

    data = pd.read_csv('train.csv',encoding = 'ISO-8859-1')
    data = data[(data['compliance'] == 0) | (data['compliance'] == 1)]
    data = data[data.agency_name != 'Neighborhood City Halls']
    data = data[data.agency_name != 'Health Department']
    test_data = pd.read_csv('test.csv')
    address =  pd.read_csv('addresses.csv')
    latlons = pd.read_csv('latlons.csv')
    address = address.set_index('address').join(latlons.set_index('address'), how='left')
    data = data.set_index('ticket_id').join(address.set_index('ticket_id'))
    target = data['compliance']
    feature = data.drop('compliance', axis = 1)
    feature['time_gap'] = feature.apply(lambda row: time_gap(row['hearing_date'], row['ticket_issued_date'
]), axis=1)
    test_data['time_gap'] = test_data.apply(lambda row: time_gap(row['hearing_date'], row['ticket_issued_d
ate']), axis=1)
    test_data = test_data.set_index('ticket_id').join(address.set_index('ticket_id'))
    test_data = test_data.drop(['violator_name', 'zip_code', 'country','city','state',
                            'inspector_name', 'violation_street_number', 'violation_street_name',
                            'violation_zip_code', 'violation_description',
                            'mailing_address_str_number', 'mailing_address_str_name',
                            'non_us_str_code',
                            'ticket_issued_date', 'hearing_date', 'grafitti_status','disposition','violation
_code'], axis = 1)
    test = list(test_data.columns)
    train = list(feature.columns)
    for column in train:
        if column not in test:
            feature = feature.drop(column, axis = 1)
    feature = pd.get_dummies(feature)
    test_data = pd.get_dummies(test_data)
    feature.lat.fillna(method='pad', inplace=True)
    feature.lon.fillna(method='pad', inplace=True)
    test_data.lat.fillna(method='pad', inplace=True)
    test_data.lon.fillna(method='pad', inplace=True)

    from sklearn.ensemble import ExtraTreesClassifier
    from sklearn.datasets import load_iris
    from sklearn.feature_selection import SelectFromModel

    clf = ExtraTreesClassifier()
    clf = clf.fit(feature, target)
    model = SelectFromModel(clf, prefit=True)
    feature = model.transform(feature)
    test_data = model.transform(test_data)

    sc = MinMaxScaler()
    feature = sc.fit_transform(feature)

    X_train,X_test, y_train, y_test = train_test_split(feature, target, test_size = 0.3, random_state = 0)
    forest = RandomForestClassifier(n_estimators = 400, class_weight ='balanced', max_depth = 7)

    forest.fit(X_train,y_train)
    test_data = sc.fit_transform(test_data)
    compliance = forest.predict_proba(test_data)[:,1]
    test_df = pd.read_csv('test.csv', encoding = "ISO-8859-1")
    test_df['compliance'] = compliance
    test_df.set_index('ticket_id', inplace=True)

    return test_df.compliance
```

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