Predicting_Property_Maintenance_Fines

March 16, 2023

0.1 Project: Understanding and Predicting Property Maintenance Fines

0.2 Table of Contents

Introduction

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?

This project is focused on understanding when and why a resident might fail to comply with a blight ticket by training a model to predict blight ticket compliance in Detroit using `readonly/train.csv`. Using this model, return the probability that each corresponding ticket from `readonly/test.csv` will be paid.

0.2.1 Dataset Description

This project is based on a data challenge from the Michigan Data Science Team (MDST)who partnered with the City of Detroit to help solve one of the most pressing blight problem facing Detroit.

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 date, 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.

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 violatio
```

```
mailing_address_str_number, mailing_address_str_name, city, state, zip_code, non_us_str_code, c
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
```

0.3 Evaluation

The predictions give the probability that the corresponding blight ticket will be paid on time. The evaluation metric for this project is the Area Under the ROC Curve (AUC). Model gives AUC score of above 0.75

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
```

Data Wrangling

[133]:
```
import pandas as pd, numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load the data files
train = pd.read_csv('train.
↪csv',encoding='ISO-8859-1',low_memory=False,parse_dates=['ticket_issued_date',␣
↪'hearing_date','payment_date'])
test  = pd.read_csv('test.csv',encoding='ISO-8859-1',low_memory=False,␣
↪parse_dates=['ticket_issued_date', 'hearing_date'])
address = pd.read_csv('addresses.csv',encoding='ISO-8859-1',low_memory=False)
coord   = pd.read_csv('latlons.csv',encoding='ISO-8859-1',low_memory=False)
```

Visually examine tables

[110]:
```
train.sample(3)
```

[110]:
```
        ticket_id                                       agency_name  \
126148     153633  Buildings, Safety Engineering & Env Department
235889     270124  Buildings, Safety Engineering & Env Department
90156      115317  Buildings, Safety Engineering & Env Department

        inspector_name         violator_name  violation_street_number  \
126148  Samaan, Neil J        HUTCHINS, OTTO                  19304.0
235889  Samaan, Neil J     BISZCZANIK, MAREK                   7425.0
90156   Samaan, Neil J  MISSIONARY, EMMANUEL                   1271.0

       violation_street_name  violation_zip_code  mailing_address_str_number  \
126148                HOOVER                 NaN                      1414.0
235889               DAVISON                 NaN                      6876.0
90156             OAKMAN BLVD                 NaN                     21143.0

       mailing_address_str_name       city  … clean_up_cost judgment_amount  \
126148                   PO BOX     WARREN  …           0.0           305.0
235889             MEADOWLAKE RD  BMFD TWP  …           0.0           305.0
90156                    PO BOX    DETROIT  …           0.0           305.0

       payment_amount balance_due payment_date       payment_status  \
126148            0.0       305.0          NaT  NO PAYMENT APPLIED
235889            0.0       305.0          NaT  NO PAYMENT APPLIED
90156             0.0       305.0          NaT  NO PAYMENT APPLIED

       collection_status grafitti_status            compliance_detail  \
126148     IN COLLECTION             NaN  non-compliant by no payment
235889     IN COLLECTION             NaN  non-compliant by no payment
90156      IN COLLECTION             NaN  non-compliant by no payment

        compliance
126148         0.0
235889         0.0
90156          0.0
```

```
[3 rows x 34 columns]
```

[111]:
```
train.shape
```

[111]:
```
(250306, 34)
```

[112]:
```
address.sample(3)
```

[112]:
```
        ticket_id                    address
152790     182446       1975 webb, Detroit MI
85121      110178   5418 iroquois, Detroit MI
163791     193718   2082 vinewood, Detroit MI
```

[113]:
```
coord.sample(3)
```

[113]:
```
                             address        lat        lon
30389        4867 avery, Detroit MI  42.351081 -83.081252
115680      1155 lenore, Detroit MI  42.421118 -83.281344
106383    1524 military, Detroit MI  42.313517 -83.104955
```

[114]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250306 entries, 0 to 250305
Data columns (total 34 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   ticket_id                   250306 non-null  int64
 1   agency_name                 250306 non-null  object
 2   inspector_name              250306 non-null  object
 3   violator_name               250272 non-null  object
 4   violation_street_number     250306 non-null  float64
 5   violation_street_name       250306 non-null  object
 6   violation_zip_code          0 non-null       float64
 7   mailing_address_str_number  246704 non-null  float64
 8   mailing_address_str_name    250302 non-null  object
 9   city                        250306 non-null  object
 10  state                       250213 non-null  object
 11  zip_code                    250305 non-null  object
 12  non_us_str_code             3 non-null       object
 13  country                     250306 non-null  object
 14  ticket_issued_date          250306 non-null  datetime64[ns]
 15  hearing_date                237815 non-null  datetime64[ns]
 16  violation_code              250306 non-null  object
 17  violation_description       250306 non-null  object
 18  disposition                 250306 non-null  object
```

```
 19  fine_amount                250305 non-null  float64
 20  admin_fee                  250306 non-null  float64
 21  state_fee                  250306 non-null  float64
 22  late_fee                   250306 non-null  float64
 23  discount_amount            250306 non-null  float64
 24  clean_up_cost              250306 non-null  float64
 25  judgment_amount            250306 non-null  float64
 26  payment_amount             250306 non-null  float64
 27  balance_due                250306 non-null  float64
 28  payment_date               41113 non-null   datetime64[ns]
 29  payment_status             250306 non-null  object
 30  collection_status          36897 non-null   object
 31  grafitti_status            1 non-null       object
 32  compliance_detail          250306 non-null  object
 33  compliance                 159880 non-null  float64
dtypes: datetime64[ns](3), float64(13), int64(1), object(17)
memory usage: 64.9+ MB
```

[115]:
```
train.isnull().sum()
```

[115]:
```
ticket_id                          0
agency_name                        0
inspector_name                     0
violator_name                     34
violation_street_number            0
violation_street_name              0
violation_zip_code            250306
mailing_address_str_number      3602
mailing_address_str_name           4
city                               0
state                             93
zip_code                           1
non_us_str_code               250303
country                            0
ticket_issued_date                 0
hearing_date                   12491
violation_code                     0
violation_description              0
disposition                        0
fine_amount                        1
admin_fee                          0
state_fee                          0
late_fee                           0
discount_amount                    0
clean_up_cost                      0
judgment_amount                    0
payment_amount                     0
```

```
balance_due                        0
payment_date                  209193
payment_status                     0
collection_status             213409
grafitti_status               250305
compliance_detail                  0
compliance                     90426
dtype: int64
```

0.3.1 Strategize wrangling path by defining cleaning process on `train` table.

Quality

- Missing demographic information (payment_date, collection_status, grafitti_status, compliance, violator_name, violation_zip_code, mailing_address_str_number, mailing_address_str_name, state, zip_code, non_us_str_code, hearing_date contact columns) ***(can't clean yet)***

Tidiness

- Columns with all entries being zero should be removed.
- Columns with the same values should be removed as they are uncorrelated with target variable.
- Columns with total unique values less than 10% of entries (<250) should be converted into categorical data to reduce memory usage to reduce memory usage.
- Remove columns with missing value % of more than 50%.
- Join the address table to train and test tables to expand features.
- With address joined, now remove features that can be replaced with address, such as :

```
['violator_name','violation_street_number', 'violation_street_name','mailing_address_str_number
'mailing_address_str_name','state', 'zip_code', 'country','address','city']
```

- Reduce the features even further, by suming the amount payables into one.
- drop missing values of ['lat','lon','total_amt_pay'] from the train dataset
- Replace ticket issue data and the hearing date with the time gap between them.
- Now remove not too important featured and make strinig features from string categories

```
['inspector_name', 'violation_code','violation_description',  'payment_amount',
'balance_due','payment_status',  'compliance_detail']
```

- taking only non-NaN values for training
- trime the train data to have only the columns available in the test data

[116]:
```
# now we remove columns and rows with all entries being EMPTY
train.dropna(how='all',axis=1, inplace=True)
train.dropna(how='all',axis=0, inplace=True)
```

[117]:
```
# Remove columns with the same values they are independent/non-correlated to/
 ↪from target values
independent = []
```

```
for i in range(len(train.columns)):
    if len(train[train.columns[i]].unique())==1:
        independent.append(train.columns[i])

train.drop(independent,axis=1,inplace=True)
test.drop(independent,axis=1,inplace=True)
```

[118]:
```
# we see that there are a lot of columns with total unique values less than 250.
↪Thus we can convert them into categorical data to reduce memory usage
# to reduce memory usage we convert columns with < than 250 entries to␣
↪categorical data

for i in range(len(train.columns)):
    if len(train[train.columns[i]].unique())<250:
        train[train.columns[i]] = train[train.columns[i]].astype('category')
```

[119]:
```
# now lets see the missing number ratio in the data set
total_null = train.isnull().sum().sort_values(ascending=False)
per        = train.isnull().count().sort_values(ascending=False)
```

[120]:
```
# now i remove columns with missing value percentage of more than 50%

high_mssing_data = pd.concat([total_null,total_null/per],␣
↪keys=['Total_nulls','percentage_nulls'],axis=1)
high_missing_values = high_mssing_data[high_mssing_data['percentage_nulls']>0.
↪5].index
train.drop(high_missing_values,axis=1,inplace=True)
```

[121]:
```
# Now we join the address to train and test data
address = address.merge(coord,how='inner',left_on='address',right_on='address')
train = train.
↪merge(address,how='left',left_on='ticket_id',right_on='ticket_id')\
                .set_index('ticket_id')
test  = test.merge(address,how='left',left_on='ticket_id',right_on='ticket_id')\
                .set_index('ticket_id')
```

[122]:
```
# now we reduce the features that can be replaced by the lat and lon
latlon_replaced = ['violator_name',
       'violation_street_number', 'violation_street_name',
       'mailing_address_str_number', 'mailing_address_str_name',
       'state', 'zip_code', 'country','address','city']
train.drop(latlon_replaced, axis=1,inplace=True)
```

[123]:
```
# Now we reduce the features even further, by suming the amount payables into␣
↪one
```

```
train['total_amt_pay'] =␣
  ↪train[['fine_amount','admin_fee','state_fee','late_fee']].sum(axis=1).
  ↪subtract(train['discount_amount'].astype(np.float64))
test['total_amt_pay']  =␣
  ↪test[['fine_amount','admin_fee','state_fee','late_fee']].sum(axis=1).
  ↪subtract(test['discount_amount'].astype(np.float64))
drop_payments =␣
  ↪['fine_amount','admin_fee','state_fee','late_fee','discount_amount']
train.drop(drop_payments,axis=1, inplace=True)
```

[124]:
```
# drop missing values of ['lat','lon','total_amt_pay'] from the train dataset␣
  ↪but since its not allowed in the test set,we replace it with the mean
train.dropna(subset = ['lat','lon','total_amt_pay'],inplace=True)
test['lat'].fillna(test.lat.mean(),inplace=True)
test['lon'].fillna(test.lon.mean(),inplace=True)
```

[125]:
```
# Now we find the time gap between the ticket issue data and the hearing date
train['time_delta'] = (train['hearing_date'] - train['ticket_issued_date']).dt.
  ↪days
test['time_delta']  = (test['hearing_date'] - test['ticket_issued_date']).dt.
  ↪days
drop_timedelta = ['hearing_date','ticket_issued_date']
train.drop(drop_timedelta,axis=1, inplace=True)
test.drop(drop_timedelta,axis=1, inplace=True)
```

[126]:
```
# Replace the missing values in the time delta column with the mode
train['time_delta'].fillna(73, inplace=True)
test['time_delta'].fillna(73,inplace=True)
```

[127]:
```
# Now remove not too important featured and make strinig features from string␣
  ↪categories 'disposition','agancy_name'
further_drop = ['inspector_name', 'violation_code','violation_description',
                'payment_amount', 'balance_due','payment_status',
                'compliance_detail']

train.drop(further_drop,axis=1, inplace=True)
string_features = ['disposition','agency_name']
train = pd.get_dummies(train,columns = string_features,drop_first=True)
test = pd.get_dummies(test,columns = string_features,drop_first=True)
```

[128]:
```
# taking only non-NaN values for training
train = train[( (train['compliance']==0) | (train['compliance']==1) )]
```

[129]:
```
# trime the train data to have only the columns available in the test data
y = train['compliance']
X = train.drop('compliance',axis=1)
```

```
train_feature_set = set(X)
for feature in set(X):
    if feature not in test:
        train_feature_set.remove(feature)

train_features = list(train_feature_set)
X_train = X[train_features]
test    = test[train_features]

# X_train, y, test,
```

Model selection, training and evalution

[53]:
```
from sklearn.model_selection import train_test_split
 ↪       ,GridSearchCV
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.dummy import DummyClassifier
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier\
                            ,RandomForestRegressor\
                            ,GradientBoostingClassifier
import time
```

[54]:
```
# divide train test split for model selection
x_train,x_test,y_train,y_test= train_test_split(X_train,y,test_size=0.
 ↪4,random_state=40)
X_train.shape,y.shape
```

[54]:
```
((159878, 11), (159878,))
```

[61]:
```
# implement dummy classifier as base model, using most frequent data as
 ↪prediction
start = time.time()
dummy_clf = DummyClassifier(strategy='most_frequent').fit(x_train,y_train)
y_pred = dummy_clf.predict(x_test)
print('Runtime: {} \nROC_score : {}'.format(time.time()- start,\
                                      roc_auc_score(y_test,y_pred)))
```

```
Runtime: 0.014135837554931641
ROC_score : 0.5
```

```
[65]: # implementing KNN classifier
start = time.time()
KN_clf = KNeighborsRegressor()
param_values = {'n_neighbors':[1,3,5,7,9]}
grid_clf = GridSearchCV(KN_clf, param_grid=param_values,scoring='roc_auc').
 ↪fit(x_train,y_train)
print('Best parameter is: {}\nBest ROC_score is: {}'.format(grid_clf.
 ↪best_params_,grid_clf.best_score_))
print('Runtime: {}'.format(time.time()-start))
```

```
Best parameter is: {'n_neighbors': 9}
Best ROC_score is: 0.7494481978035308
Runtime: 29.89334988594055
```

```
[68]: # implenting Ridge classifier
start = time.time()
R_clf = RidgeClassifier()
R_clf.get_params().keys()
param_values = {'alpha':[0, 1, 10, 20, 50, 100, 1000]}
grid_clf = GridSearchCV(R_clf, param_grid=param_values,scoring='roc_auc').
 ↪fit(x_train,y_train)
print('Best parameter is {}\nBest ROC_score is {}'.format(grid_clf.
 ↪best_params_,grid_clf.best_score_))
print('Runtime: {}'.format(time.time()-start))
```

```
/Users/air/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_ridge.py:157: LinAlgWarning: Ill-conditioned
matrix (rcond=1.03558e-22): result may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
/Users/air/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_ridge.py:157: LinAlgWarning: Ill-conditioned
matrix (rcond=9.24063e-23): result may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
```

```
Best parameter is {'alpha': 0}
Best ROC_score is 0.7723439974165995
Runtime: 3.734174966812134
```

```
/Users/air/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_ridge.py:157: LinAlgWarning: Ill-conditioned
matrix (rcond=5.47116e-23): result may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
```

```
[72]: # implenting random forest regressor
start = time.time()
RF_clf = RandomForestRegressor()
param_values = {'max_depth':[1,3,5,7,9,11,13,15,17,18,21]}
# X_train = MinMaxScaler().fit_transform(X_train)
```

```
# # testt = MinMaxScaler().fit_transform(test)
grid_clf = GridSearchCV(RF_clf, param_grid=param_values,scoring='roc_auc').
 ↪fit(x_train,y_train)
print('Best parameter is {}\nBest ROC_score is {}'.format(grid_clf.
 ↪best_params_,grid_clf.best_score_))
print('Runtime: {}'.format(time.time()-start))
```

```
Best parameter is {'max_depth': 15}
Best ROC_score is 0.8197865894627888
Runtime: 1321.5681660175323
```

[73]:
```
# implementing support vector machine classifier
start = time.time()
SV = SVC(kernel = 'rbf',C = 0.01)
param_value = {'gamma': [0.01,0.1,1,10,100,]}
grid_clf = GridSearchCV(SV,param_grid= param_value ,scoring = 'roc_auc').
 ↪fit(x_train,y_train)
print('Best parameter is {}\nBest ROC_score is {}'.format(grid_clf.
 ↪best_params_,grid_clf.best_score_))
print('Runtime: {}'.format(time.time()-start))
```

```
Best parameter is {'gamma': 100}
Best ROC_score is 0.6964877516589298
Runtime: 4802.026435136795
```

Random forest regressor is selected

[130]:
```
def blight_model(X_train, y, test):
    #import necessary models to train the data
    from sklearn.ensemble import RandomForestRegressor

    RF_clf = RandomForestRegressor(max_depth=6).fit(X_train,y)
    y_pred = RF_clf.predict(test)
    test['compliance'] = y_pred
    return test.compliance
```

[131]:
```
blight_model(X_train, y, test)
```

[131]:
```
ticket_id
284932    0.063016
285362    0.013487
285361    0.068180
285338    0.083102
285346    0.087070
            …
376496    0.013312
376497    0.013312
376499    0.070316
```

```
376500    0.070316
369851    0.399647
Name: compliance, Length: 61001, dtype: float64
```

[ ]:

```

```