State Farm Project Model 1

July 29, 2019

1 State Farm Classification Project: Model 1

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_validate, train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, r2
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn import metrics
```

In [2]:
```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
```

In [3]:
```
train = pd.read_csv("exercise_02_train.csv")
```

In [4]:
```
train.head()
```

Out[4]:
```
          x0         x1         x2        x3         x4         x5        x6  \
0   0.198560  74.425320  67.627745 -3.095111  -6.822327  19.048071 -0.362378
1 -29.662621  24.320711 -48.205182  1.430339  -6.552206   4.263074  6.551412
2  15.493759 -66.160459  50.512903 -2.265792  14.428578   2.509323 -6.707536
3 -19.837651  33.210943  53.405563  1.079462  11.364251  -1.064581  9.308857
4  11.896655 -26.717872 -17.758176  1.692017  21.553537  -5.852097 -0.857435

          x7         x8        x9 ...       x91       x92      x93       x94  \
0 -10.699174 -22.699791 -1.561262 ...  0.800948  1.553846     asia -1.093926
1   4.265483   1.245095  2.246814 ...  2.031707  7.544422     asia -3.659541
2   3.820842 -11.100833 -1.459825 ... -0.992474  1.385799  america  1.299144
3   9.266076  14.552959 -2.012755 ... -1.157845  6.036804     asia  0.521396
4  -2.186940  18.075272 -1.404618 ... -3.045511 -1.719337     asia  1.526071

         x95        x96        x97        x98       x99  y
0  16.202557  26.238591  -2.125570   9.644466  1.237667  0
1  29.674259 -15.141647 -36.030599   5.820376  1.952183  1
2  33.018090 -19.914894  26.212736   2.372690  0.558988  1
3   9.664095 -27.197636  19.221130  13.382712  0.214462  0
4 -25.608326  33.383803  -5.703269 -11.023730 -1.191319  0

[5 rows x 101 columns]
```

```
In [5]: def unique(list1,getList = False):

            # insert the list to the set
            list_set = set(list1)
            # convert the set to the list
            unique_list = (list(list_set))
            for x in unique_list:
                print(x)
            if getList == True:
                return unique_list

        unique(train.dtypes)
```

```
float64
int64
object
```

```
In [6]: predictCols = list(train)
        predictCols.remove('y')
```

```
In [7]: for col in predictCols:
            if train[col].dtype in [np.float64,np.int64]:
                #print(col)
                train[col].fillna(train[col].mean(skipna = True), inplace=True)
```

```
In [8]: # Ensure no remaining na's
        numericCols = train.select_dtypes(include='number').columns
        naVals = train[numericCols].isna().sum().sort_values()
        naVals.sum()
```

```
Out[8]: 0
```

```
In [9]: objectCols = train.select_dtypes(include='object').columns
        print(objectCols)
```

```
Index(['x34', 'x35', 'x41', 'x45', 'x68', 'x93'], dtype='object')
```

```
In [10]: train.x34.fillna(train.x34.mode()[0], inplace=True)
         uniqueX34_train = unique(train['x34'],True)
```

```
Toyota
bmw
volkswagon
tesla
ford
Honda
chrystler
```

```
mercades
chevrolet
nissan
```

```
In [11]: train.x35.replace(['thurday', 'thur'], ['thursday','thursday'], inplace=True)
         train.x35.replace(['wed'], ['wednesday'], inplace=True)
         train.x35.replace(['fri'], ['friday'], inplace=True)
         train.x35.fillna(train.x35.mode()[0], inplace=True)
         unique(train['x35'])
```

```
thursday
tuesday
monday
friday
wednesday
```

```
In [12]: # Convert currency column to float, remove nan's
         train['x41'] = train['x41'].astype(str)
         train['x41'] = train['x41'].map(lambda x: x.lstrip('$'))
         train['x41'] = train['x41'].astype(np.float16)
         train['x41'].fillna(0, inplace=True) # probably safer to replace nan's with 0, not me

         print(train['x41'].isna().sum())
```

```
0
```

```
In [13]: # Convert percentage column to float, remove nan's
         train['x45'] = train['x45'].astype(str)
         train['x45'] = train['x45'].map(lambda x: x.rstrip('%'))
         train['x45'] = train['x45'].astype(np.float16)
         train['x45'].fillna(train['x45'].mean(skipna = True), inplace=True) # since very few

         print(train['x41'].isna().sum())
```

```
0
```

```
In [14]: # Month Column
         train.x68.replace(['Dev'], ['Dec'], inplace=True) # because I'm OCD
         train.x68.replace(['sept.'], ['Sep'], inplace=True)
         train.x68.replace(['January'], ['Jan'], inplace=True)
         train.x68.replace(['July'], ['Jul'], inplace=True)
         train.x68.fillna(train.x68.mode()[0], inplace=True)

         unique(train['x68'])
```

```
Jun
May
Feb
Aug
Oct
Mar
Jan
Nov
Apr
Sep
Dec
Jul
```

In [15]:
```python
# Region
train.x93.replace(['euorpe'], ['europe'], inplace=True)
train = train[pd.isna(train['x93']) == False]
print(train['x93'].isna().sum())
# Region seems significant, and there's only 7 NA's, so remove rows with this as NA
```

```
0
```

In [16]:
```python
# Check if target has na's
print(train['y'].isna().sum())
```

```
0
```

In [17]:
```python
train = pd.get_dummies(train)
```

In [18]:
```python
# Ensure we converted all non-numeric columns to numeric
train.select_dtypes(include='object').columns
```

Out[18]:
```
Index([], dtype='object')
```

In [19]:
```python
train.describe()
```

Out[19]:
```
                 x0            x1            x2            x3            x4  \
count  39993.000000  39993.000000  39993.000000  39993.000000  39993.000000
mean       3.447752     -7.788416      1.704644     -0.072832      0.121980
std       16.245334     37.012224     38.382930      1.503022     16.289301
min      -60.113902   -157.341119   -163.339956     -6.276969    -61.632319
25%       -7.595295    -32.731869    -24.141082     -1.087780    -10.896141
50%        3.446322     -7.987507      1.959477     -0.062721      0.105307
75%       14.266326     16.848201     27.511371      0.940330     11.076726
max       75.311659    153.469221    154.051060      5.837559     65.949709

                 x5            x6            x7            x8            x9  \
```

```
count  39993.000000  39993.000000  39993.000000  39993.000000  39993.000000
mean      -0.607009      0.035852     -0.052430     -2.911144     -0.024524
std       15.583132      9.040667      6.952184     13.148182      2.939696
min      -62.808995    -35.060656    -26.736717    -53.735586    -11.497395
25%      -11.181510     -6.089227     -4.746572    -11.722590     -2.003827
50%       -0.576660      0.044975     -0.037833     -2.940961     -0.054184
75%        9.954957      6.100325      4.636585      5.857648      1.954809
max       63.424046     45.053946     34.267792     66.936936     11.271939

                  ...       x68_Jul       x68_Jun       x68_Mar       x68_May  \
count             ...  39993.000000  39993.000000  39993.000000  39993.000000
mean              ...      0.277199      0.231516      0.010777      0.119221
std               ...      0.447621      0.421806      0.103252      0.324052
min               ...      0.000000      0.000000      0.000000      0.000000
25%               ...      0.000000      0.000000      0.000000      0.000000
50%               ...      0.000000      0.000000      0.000000      0.000000
75%               ...      1.000000      0.000000      0.000000      0.000000
max               ...      1.000000      1.000000      1.000000      1.000000

             x68_Nov       x68_Oct       x68_Sep  x93_america      x93_asia  \
count   39993.000000  39993.000000  39993.000000  39993.000000  39993.000000
mean        0.003776      0.022604      0.087140      0.078289      0.885555
std         0.061331      0.148639      0.282044      0.268629      0.318355
min         0.000000      0.000000      0.000000      0.000000      0.000000
25%         0.000000      0.000000      0.000000      0.000000      1.000000
50%         0.000000      0.000000      0.000000      0.000000      1.000000
75%         0.000000      0.000000      0.000000      0.000000      1.000000
max         1.000000      1.000000      1.000000      1.000000      1.000000

          x93_europe
count   39993.000000
mean        0.036156
std         0.186681
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 127 columns]
```

1.1 Now check class imbalance

In [20]:
```python
x = train.y.value_counts()
print(x)

print("% oF Training Set with Positives: " + "{0:.0%}".format(x[1] / (x[0] + x[1])))
print("% oF Training Set with Negatives: " + "{0:.0%}".format(x[0] / (x[0] + x[1])))
```

```
0    31851
1     8142
Name: y, dtype: int64
% oF Training Set with Positives: 20%
% oF Training Set with Negatives: 80%
```

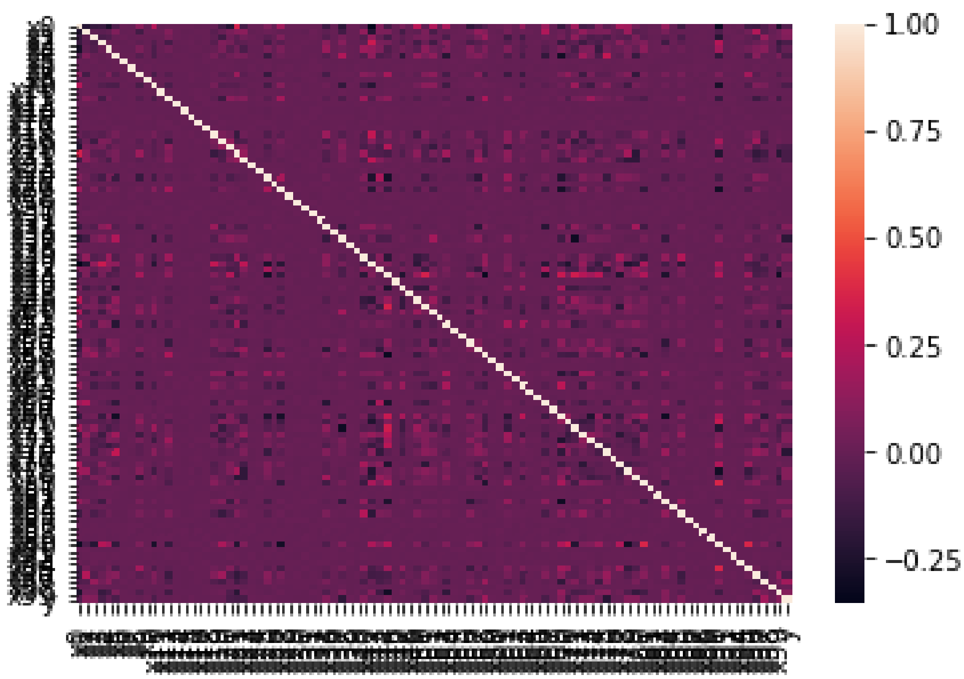
1.1.1 This class imbalance is not too bad, so we don't need to do resampling...

In [ ]:

1.2 Now check correlation

In [21]:
```
import seaborn as sns
%matplotlib inline

# calculate the correlation matrix
corr = train[numericCols].corr()

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x24fec000b38>



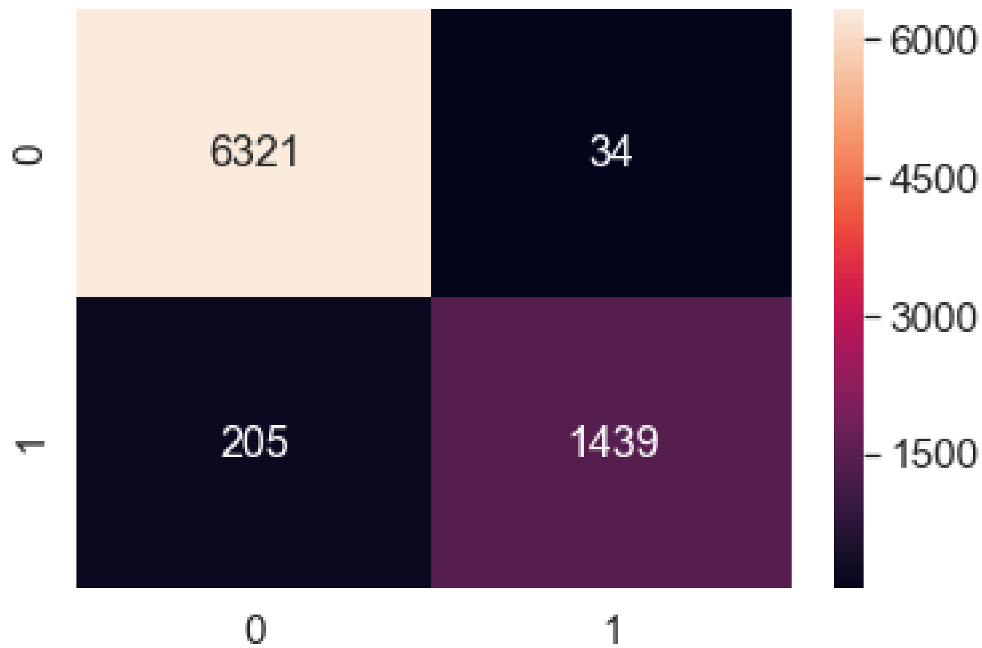
```
In [22]: # Attempt to find columns with high-correlation, and remove if necessary
         for col in numericCols:
             #print("\n\n" + col)
             q = corr[col].sort_values(ascending = False)
             q = q.drop(col)
             q = q[abs(q) >= 0.5]
             perfectNames = q.index.values
             if len(q) > 0:
                 print(col)
```

1.3 Clearly, no columns in the data set are highly correlated, so no need to remove.

2 Now can send to XGBoost

```
In [23]: # Train test split
         y = train.y
         train = train.drop(['y'], axis=1)

In [24]: X_train, X_test, y_train, y_test = train_test_split(train, y, test_size=.2, random_sta

In [25]: dtrain = xgb.DMatrix(X_train)
         dtest = xgb.DMatrix(X_test)

In [26]: # First XGboost attempt, leaving most parameters as default
         model = xgb.XGBClassifier(booster='gbtree',objective ='binary:logistic',max_depth = 5
                                   min_child_weight = 1,n_estimators = 500,seed = 1,n_jobs = 1:

         model.fit(X_train,y_train)

Out[26]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
                max_delta_step=0, max_depth=5, min_child_weight=1, missing=None,
                n_estimators=500, n_jobs=12, nthread=None,
                objective='binary:logistic', random_state=0, reg_alpha=0,
                reg_lambda=1, scale_pos_weight=1, seed=1, silent=None, subsample=1,
                verbosity=1)

In [27]: pred_train = model.predict(X_train)
         pred_test = model.predict(X_test)

In [28]: # evaluate predictions
         test_accuracy = accuracy_score(y_test, pred_test)
         print("Test Accuracy: %.2f%%" % (test_accuracy * 100.0))

Test Accuracy: 97.01%


In [29]: cm = confusion_matrix(y_test, pred_test)

         sns.set(font_scale=1.4)#for label size
         sns.heatmap(cm, annot=True,fmt='g',annot_kws={"size": 16})# font size
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x24febe589b0>
```



2.0.1 Many more false negatives than false postives...probably due to the class imbalance 0's to 1's in the training set heavily weighted toward 0's (80%)

```
In [ ]:
```

```
In [46]: # Tune max_depth and min_child_weight, since these have the highest impact on the mod
         param_test1 = {
          'max_depth':range(5,12,1),
          'min_child_weight':range(1,6,1)
         }
         gsearch1 = GridSearchCV(estimator = XGBClassifier(booster='gbtree',objective ='binary
                                   n_estimators = 500,seed = 1,n_jobs = 12),
          param_grid = param_test1, scoring='roc_auc',n_jobs=4,iid=False, cv=5)
         gsearch1.fit(X_train,y_train)
```

```
Out[46]: GridSearchCV(cv=5, error_score='raise-deprecating',
                estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
                max_delta_step=0, max_depth=10, min_child_weight=1, missing=None,
                n_estimators=500, n_jobs=4, nthread=None,
                objective='binary:logistic', random_state=0, reg_alpha=0,
                reg_lambda=1, scale_pos_weight=1, seed=1, silent=None, subsample=1,
                verbosity=1),
```

```
             fit_params=None, iid=False, n_jobs=4,
             param_grid={'max_depth': range(5, 12), 'min_child_weight': range(1, 6)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring='roc_auc', verbose=0)
```

In [47]:
```
gsearch1.best_params_, gsearch1.best_score_
```

Out[47]:
```
({'max_depth': 8, 'min_child_weight': 4}, 0.9892715407729563)
```

In [44]:
```
# Tune max_depth and min_child_weight, since these have the highest impact on the mod
param_test2 = {
 'n_estimators':range(100,600,100),
 'learning_rate':[0.07,0.08,0.09,0.1,0.11,0.12,0.13]
}

gsearch2 = GridSearchCV(estimator = XGBClassifier(booster='gbtree',objective ='binary
                          min_child_weight = 1,n_estimators = 500,seed = 1,n_jobs = 4)
 param_grid = param_test2, scoring='roc_auc',n_jobs=4,iid=False, cv=5)

gsearch2.fit(X_train,y_train)
```

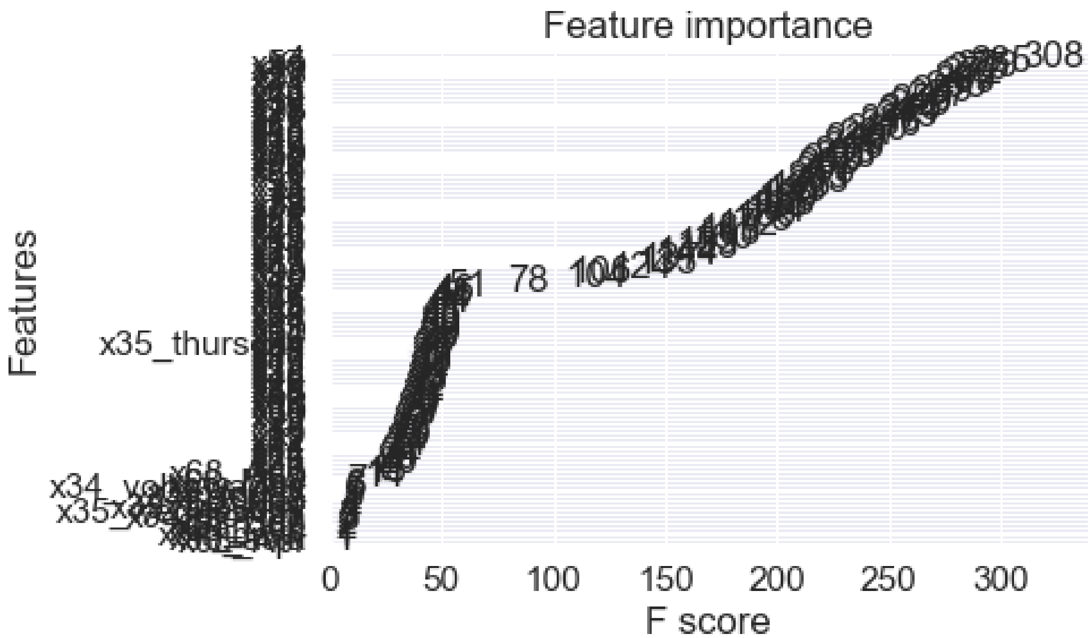
Out[44]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
             max_delta_step=0, max_depth=7, min_child_weight=1, missing=None,
             n_estimators=500, n_jobs=4, nthread=None,
             objective='binary:logistic', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, seed=1, silent=None, subsample=1,
             verbosity=1),
             fit_params=None, iid=False, n_jobs=4,
             param_grid={'n_estimators': range(100, 600, 100), 'learning_rate': [0.07, 0.08
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring='roc_auc', verbose=0)
```

In [45]:
```
gsearch2.best_params_, gsearch2.best_score_
```

Out[45]:
```
({'learning_rate': 0.1, 'n_estimators': 500}, 0.9892144265557844)
```

2.1 Now attempt to understand feature importance

In [30]:
```
xgb.plot_importance(model)
plt.rcParams['figure.figsize'] = [5, 5]
plt.show()
```




```
In [31]: columns = model.get_booster().get_score(importance_type="gain")
```

```
In [32]: dframe = pd.DataFrame([columns])
         dframe = dframe.T
         dframe.index.names = ['Feature Name']
         dframe.columns = ['Importance']
         dframe.sort_values(['Importance'],ascending = False)
```

Out[32]:

Feature Name	Importance
x75	34.724857
x97	32.805042
x37	28.269208
x35_thursday	24.789958
x58	24.267436
x41	23.591614
x53	23.122496
x99	21.670210
x66	21.234005
x83	18.777649
x51	18.483428
x12	16.935363
x44	16.617990
x50	16.508720
x1	16.345584
x63	16.297937

```
x40          16.162018
x78          15.747694
x21          15.613956
x5           15.381603
x3           15.074353
x10          14.805315
x96          13.945939
x70          13.843800
x2           13.234008
x71          13.082727
x45          12.516872
x56          11.985151
x19          11.566486
x72          11.439885
...                ...
x6            1.786068
x89           1.749735
x98           1.748289
x91           1.741379
x65           1.719907
x81           1.716288
x9            1.670220
x46           1.648257
x62           1.643576
x31           1.638202
x88           1.629320
x87           1.565456
x67           1.532457
x14           1.532142
x17           1.527966
x52           1.520013
x54           1.495932
x36           1.477934
x29           1.394242
x13           1.390530
x60           1.369393
x34_Toyota    1.272307
x32           1.248057
x68_Dec       1.236412
x59           1.182015
x93_asia      1.165440
x34_Honda     1.059672
x35_friday    1.046746
x68_Mar       1.027684
x34_bmw       0.983792

[113 rows x 1 columns]
```

```
In [37]: from numpy import sort
         from sklearn.feature_selection import SelectFromModel

         thresholds = sort(model.feature_importances_)
         for thresh in thresholds:
             # select features using threshold
             selection = SelectFromModel(model, threshold=thresh, prefit=True)
             select_X_train = selection.transform(X_train)
             # train model
             selection_model = XGBClassifier(booster='gbtree',objective = 'binary:logistic',ma
                                  min_child_weight = 1,n_estimators = 500,seed = 1,n_jobs = 1
                                  learning_rate = 0.1)
             selection_model.fit(select_X_train, y_train)
             # eval model
             select_X_test = selection.transform(X_test)
             y_pred = selection_model.predict(select_X_test)
             predictions = [round(value) for value in y_pred]
             accuracy = accuracy_score(y_test, predictions)
             print("Thresh=%.3f, n=%d, Accuracy: %.2f%%" % (thresh, select_X_train.shape[1], a
```

```
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.000, n=126, Accuracy: 97.90%
Thresh=0.001, n=113, Accuracy: 97.91%
Thresh=0.001, n=112, Accuracy: 97.85%
Thresh=0.001, n=111, Accuracy: 97.90%
Thresh=0.001, n=110, Accuracy: 97.90%
Thresh=0.001, n=109, Accuracy: 97.81%
Thresh=0.001, n=108, Accuracy: 97.72%
Thresh=0.001, n=107, Accuracy: 98.05%
Thresh=0.001, n=106, Accuracy: 98.05%
Thresh=0.001, n=105, Accuracy: 97.84%
Thresh=0.002, n=104, Accuracy: 97.90%
Thresh=0.002, n=103, Accuracy: 97.86%
Thresh=0.002, n=102, Accuracy: 97.90%
Thresh=0.002, n=101, Accuracy: 97.87%
Thresh=0.002, n=100, Accuracy: 97.85%
Thresh=0.002, n=99, Accuracy: 97.96%
```

```
Thresh=0.002, n=98, Accuracy: 97.90%
Thresh=0.002, n=97, Accuracy: 98.07%
Thresh=0.002, n=96, Accuracy: 97.91%
Thresh=0.002, n=95, Accuracy: 97.79%
Thresh=0.002, n=94, Accuracy: 97.82%
Thresh=0.002, n=93, Accuracy: 97.77%
Thresh=0.002, n=92, Accuracy: 97.85%
Thresh=0.002, n=91, Accuracy: 97.81%
Thresh=0.002, n=90, Accuracy: 97.87%
Thresh=0.002, n=89, Accuracy: 97.81%
Thresh=0.002, n=88, Accuracy: 97.86%
Thresh=0.002, n=87, Accuracy: 98.00%
Thresh=0.002, n=86, Accuracy: 97.94%
Thresh=0.002, n=85, Accuracy: 97.97%
Thresh=0.002, n=84, Accuracy: 97.89%
Thresh=0.002, n=83, Accuracy: 98.00%
Thresh=0.002, n=82, Accuracy: 97.95%
Thresh=0.002, n=81, Accuracy: 97.97%
Thresh=0.002, n=80, Accuracy: 97.85%
Thresh=0.002, n=79, Accuracy: 97.96%
Thresh=0.002, n=78, Accuracy: 97.84%
Thresh=0.002, n=77, Accuracy: 97.85%
Thresh=0.002, n=76, Accuracy: 98.01%
Thresh=0.002, n=75, Accuracy: 98.05%
Thresh=0.002, n=74, Accuracy: 97.99%
Thresh=0.002, n=73, Accuracy: 97.94%
Thresh=0.002, n=72, Accuracy: 98.11%
Thresh=0.002, n=71, Accuracy: 98.12%
Thresh=0.002, n=70, Accuracy: 98.04%
Thresh=0.002, n=69, Accuracy: 98.05%
Thresh=0.002, n=68, Accuracy: 98.12%
Thresh=0.003, n=67, Accuracy: 97.99%
Thresh=0.003, n=66, Accuracy: 97.97%
Thresh=0.003, n=65, Accuracy: 98.00%
Thresh=0.003, n=64, Accuracy: 97.97%
Thresh=0.003, n=63, Accuracy: 98.19%
Thresh=0.003, n=62, Accuracy: 98.05%
Thresh=0.003, n=61, Accuracy: 98.12%
Thresh=0.003, n=60, Accuracy: 98.15%
Thresh=0.003, n=59, Accuracy: 98.12%
Thresh=0.006, n=58, Accuracy: 98.25%
Thresh=0.006, n=57, Accuracy: 97.95%
Thresh=0.006, n=56, Accuracy: 98.19%
Thresh=0.006, n=55, Accuracy: 98.02%
Thresh=0.007, n=54, Accuracy: 98.02%
Thresh=0.007, n=53, Accuracy: 97.92%
Thresh=0.007, n=52, Accuracy: 98.00%
Thresh=0.008, n=51, Accuracy: 98.01%
```

```
Thresh=0.008, n=50, Accuracy: 97.94%
Thresh=0.008, n=49, Accuracy: 97.94%
Thresh=0.009, n=48, Accuracy: 97.72%
Thresh=0.009, n=47, Accuracy: 97.59%
Thresh=0.009, n=46, Accuracy: 97.61%
Thresh=0.010, n=45, Accuracy: 97.64%
Thresh=0.010, n=44, Accuracy: 97.54%
Thresh=0.010, n=43, Accuracy: 97.34%
Thresh=0.010, n=42, Accuracy: 97.56%
Thresh=0.010, n=41, Accuracy: 97.42%
Thresh=0.010, n=40, Accuracy: 97.32%
Thresh=0.011, n=39, Accuracy: 96.94%
Thresh=0.011, n=38, Accuracy: 96.86%
Thresh=0.011, n=37, Accuracy: 96.81%
Thresh=0.012, n=36, Accuracy: 96.57%
Thresh=0.012, n=35, Accuracy: 96.44%
Thresh=0.012, n=34, Accuracy: 96.37%
Thresh=0.013, n=33, Accuracy: 96.44%
Thresh=0.013, n=32, Accuracy: 95.92%
Thresh=0.013, n=31, Accuracy: 95.85%
Thresh=0.013, n=30, Accuracy: 95.79%
Thresh=0.013, n=29, Accuracy: 95.55%
Thresh=0.014, n=28, Accuracy: 95.46%
Thresh=0.014, n=27, Accuracy: 95.12%
Thresh=0.015, n=26, Accuracy: 94.82%
Thresh=0.015, n=25, Accuracy: 94.89%
Thresh=0.016, n=24, Accuracy: 94.59%
Thresh=0.016, n=23, Accuracy: 94.25%
Thresh=0.017, n=22, Accuracy: 94.01%
Thresh=0.017, n=21, Accuracy: 93.71%
Thresh=0.017, n=20, Accuracy: 93.57%
Thresh=0.018, n=19, Accuracy: 92.95%
Thresh=0.018, n=18, Accuracy: 92.26%
Thresh=0.018, n=17, Accuracy: 91.76%
Thresh=0.019, n=16, Accuracy: 91.07%
Thresh=0.019, n=15, Accuracy: 90.44%
Thresh=0.019, n=14, Accuracy: 89.46%
Thresh=0.019, n=13, Accuracy: 88.65%
Thresh=0.019, n=12, Accuracy: 88.00%
Thresh=0.021, n=11, Accuracy: 86.91%
Thresh=0.021, n=10, Accuracy: 86.22%
Thresh=0.024, n=9, Accuracy: 85.05%
Thresh=0.025, n=8, Accuracy: 83.59%
Thresh=0.026, n=7, Accuracy: 82.34%
Thresh=0.027, n=6, Accuracy: 80.84%
Thresh=0.028, n=5, Accuracy: 80.04%
Thresh=0.028, n=4, Accuracy: 79.58%
Thresh=0.032, n=3, Accuracy: 79.42%
```

```
Thresh=0.037, n=2, Accuracy: 79.10%
Thresh=0.039, n=1, Accuracy: 79.50%
```

2.1.1 This shows that only a few features (<30) are really being used in the trees, and we can probably reduce the required dataset for this model considerably.

In [ ]:

2.2 Finally, pre-proccess the test set, and calculate the final prediction

In [33]:
```python
test = pd.read_csv("exercise_02_test.csv")
```

In [34]:
```python
len(test.index)
```

Out[34]:
```
10000
```

In [35]:
```python
predictCols = list(test)
for col in predictCols:
    if test[col].dtype in [np.float64,np.int64]:
        test[col].fillna(test[col].mean(skipna = True), inplace=True)
```

In [36]:
```python
# Ensure all values in test.34 are in train.34 - i.e. no alternate spellings
any(elem in unique(test.x34,True)  for elem in uniqueX34_train)
```

```
bmw
Toyota
volkswagon
tesla
ford
nan
Honda
chrystler
mercades
chevrolet
nissan
```

Out[36]:
```
True
```

In [37]:
```python
# Output is true, so all values in test.X34 are a subset of train.x34
```

In [38]:
```python
unique(test.x35)

test.x35.replace(['thurday', 'thur'], ['thursday','thursday'], inplace=True)
test.x35.replace(['wed'], ['wednesday'], inplace=True)
test.x35.replace(['fri'], ['friday'], inplace=True)
test.x35.fillna(test.x35.mode()[0], inplace=True)
```

```
tuesday
fri
wed
monday
thurday
friday
wednesday
thur
```

```
In [39]: # Convert currency column to float, remove nan's
         test['x41'] = test['x41'].astype(str)
         test['x41'] = test['x41'].map(lambda x: x.lstrip('$'))
         test['x41'] = test['x41'].astype(np.float16)
         test['x41'].fillna(0, inplace=True) # probably safer to replace nan's with 0, not mea
```

```
In [40]: # Convert percentage column to float, remove nan's
         test['x45'] = test['x45'].astype(str)
         test['x45'] = test['x45'].map(lambda x: x.rstrip('%'))
         test['x45'] = test['x45'].astype(np.float16)
         test['x45'].fillna(train['x45'].mean(skipna = True), inplace=True) # since very few u
```

```
In [41]: unique(test.x68)

         # Month Column
         test.x68.replace(['Dev'], ['Dec'], inplace=True) # because I'm OCD
         test.x68.replace(['sept.'], ['Sep'], inplace=True)
         test.x68.replace(['January'], ['Jan'], inplace=True)
         test.x68.replace(['July'], ['Jul'], inplace=True)
         test.x68.fillna(test.x68.mode()[0], inplace=True)
```

```
nan
Jun
May
Feb
Aug
sept.
Oct
Mar
Dev
January
July
Nov
Apr
```

```
In [43]: # Region
         test.x93.replace(['euorpe'], ['europe'], inplace=True)
         test.x93.fillna(test.x93.mode()[0], inplace=True)
         print(test['x93'].isna().sum())
```

```
0
```

```
In [44]: test = pd.get_dummies(test)
```

```
In [45]: # Ensure all columns in test are also in train after the one-hot encoding
         any(elem in list(test)  for elem in list(train))
```

```
Out[45]: True
```

2.3 Now retrain the model with the final hyperparameters using the full training set

```
In [46]: # First XGboost attempt, leaving most parameters as default
         model = xgb.XGBClassifier(booster='gbtree',objective = 'binary:logistic',max_depth = 7
                                   min_child_weight = 1,n_estimators = 500,seed = 1,n_jobs = 12
                                   learning_rate = 0.1)

         model.fit(train,y)
```

```
Out[46]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
                max_delta_step=0, max_depth=7, min_child_weight=1, missing=None,
                n_estimators=500, n_jobs=12, nthread=None,
                objective='binary:logistic', random_state=0, reg_alpha=0,
                reg_lambda=1, scale_pos_weight=1, seed=1, silent=None, subsample=1,
                verbosity=1)
```

2.4 Now generate the final test output

```
In [48]: final_y = model.predict_proba(test) # return class probabilities
```

```
In [49]: final_y = final_y[:,1] # return only the probability of the 1's class
```

```
In [50]: np.savetxt("results1.csv", final_y, delimiter = ",")
```