1 (XGBoost Classifier with OneHot Encoding)

February 17, 2020

0.0.1 Importing Relevant Libraries and Data

[14]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split,GridSearchCV,␣
 ↪cross_val_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report, confusion_matrix

from xgboost import XGBRFClassifier
```

[15]:
```
train_data = pd.read_csv('credit_card_default_train.csv')
test_data = pd.read_csv('credit_card_default_test.csv')
```

[16]:
```
cat_cols =␣
 ↪['Gender','EDUCATION_STATUS','MARITAL_STATUS','AGE','PAY_JULY','PAY_AUG','PAY_SEP','PAY_OCT
target = 'NEXT_MONTH_DEFAULT'
ID = 'Client_ID'
num_cols = [col for col in train_data.columns.tolist() if col not in cat_cols␣
 ↪+[target]+[ID]]
```

0.0.2 Feature Engineering

[17]:
```
def Paid_Due_July(row):
    if row['PAID_AMT_JULY'] == 0:
        val = row['DUE_AMT_JULY']
    else:
        val = row['DUE_AMT_JULY']/row['PAID_AMT_JULY']
    return val

def Paid_Due_Aug(row):
```

```python
    if row['PAID_AMT_AUG'] == 0:
        val = row['DUE_AMT_AUG']
    else:
        val = row['DUE_AMT_AUG']/row['PAID_AMT_AUG']
    return val

def Paid_Due_Sep(row):
    if row['PAID_AMT_SEP'] == 0:
        val = row['DUE_AMT_SEP']
    else:
        val = row['DUE_AMT_SEP']/row['PAID_AMT_SEP']
    return val

def Paid_Due_Oct(row):
    if row['PAID_AMT_OCT'] == 0:
        val = row['DUE_AMT_OCT']
    else:
        val = row['DUE_AMT_OCT']/row['PAID_AMT_OCT']
    return val

def Paid_Due_Nov(row):
    if row['PAID_AMT_NOV'] == 0:
        val = row['DUE_AMT_NOV']
    else:
        val = row['DUE_AMT_NOV']/row['PAID_AMT_NOV']
    return val

def Paid_Due_Dec(row):
    if row['PAID_AMT_DEC'] == 0:
        val = row['DUE_AMT_DEC']
    else:
        val = row['DUE_AMT_DEC']/row['PAID_AMT_DEC']
    return val
```

[18]:

```python
train_data['PAID_DUE_JULY'] = train_data.apply(Paid_Due_July, axis=1)
train_data['PAID_DUE_AUG'] = train_data.apply(Paid_Due_Aug, axis=1)
train_data['PAID_DUE_SEP'] = train_data.apply(Paid_Due_Sep, axis=1)
train_data['PAID_DUE_OCT'] = train_data.apply(Paid_Due_Oct, axis=1)
train_data['PAID_DUE_NOV'] = train_data.apply(Paid_Due_Nov, axis=1)
train_data['PAID_DUE_DEC'] = train_data.apply(Paid_Due_Dec, axis=1)

test_data['PAID_DUE_JULY'] = test_data.apply(Paid_Due_July, axis=1)
test_data['PAID_DUE_AUG'] = test_data.apply(Paid_Due_Aug, axis=1)
test_data['PAID_DUE_SEP'] = test_data.apply(Paid_Due_Sep, axis=1)
test_data['PAID_DUE_OCT'] = test_data.apply(Paid_Due_Oct, axis=1)
test_data['PAID_DUE_NOV'] = test_data.apply(Paid_Due_Nov, axis=1)
test_data['PAID_DUE_DEC'] = test_data.apply(Paid_Due_Dec, axis=1)
```

[19]:
```
train_data['PAY_TOT'] = train_data['PAY_JULY'] + train_data['PAY_AUG'] +␣
 ↪train_data['PAY_SEP'] + train_data['PAY_OCT'] + train_data['PAY_NOV'] +␣
 ↪train_data['PAY_DEC']
test_data['PAY_TOT'] = test_data['PAY_JULY'] + test_data['PAY_AUG'] +␣
 ↪test_data['PAY_SEP'] + test_data['PAY_OCT'] + test_data['PAY_NOV'] +␣
 ↪test_data['PAY_DEC']
```

[20]:
```
def isZero(row):
    if row['PAY_TOT'] ==0:
        val = 1
    else:
        val = 0
    return val
```

[21]:
```
train_data['PAY_TOT_0'] = train_data.apply(isZero, axis=1)
test_data['PAY_TOT_0'] = test_data.apply(isZero, axis=1)
```

[22]:
```
# creating instance of one-hot-encoder
enc = OneHotEncoder(handle_unknown='ignore')

# passing bridge-types-cat column (label encoded values of bridge_types)

train_df = pd.DataFrame(enc.
 ↪fit_transform(train_data[['Balance_Limit_V1','EDUCATION_STATUS','MARITAL_STATUS','AGE','Gen
 ↪toarray())

enc = OneHotEncoder(handle_unknown='ignore')

# passing bridge-types-cat column (label encoded values of bridge_types)

test_df = pd.DataFrame(enc.
 ↪fit_transform(test_data[['Balance_Limit_V1','EDUCATION_STATUS','MARITAL_STATUS','AGE','Gend
 ↪toarray())
```

[23]:
```
train_data = pd.concat([train_data,train_df],axis=1)
test_data = pd.concat([test_data,test_df],axis=1)
```

0.0.3 Model Selection and Evaluation

[24]:
```
X = train_data.
 ↪drop([target,ID,'Balance_Limit_V1','Gender','EDUCATION_STATUS','MARITAL_STATUS','AGE'],axis
y = train_data[target]

random_grid = {'n_estimators': [100,200,400],
               'learning_rate': [0.1,0.01,0.03]}
```

```python
grid = GridSearchCV(XGBRFClassifier(),random_grid,refit=True,verbose=3,cv=3)
grid.fit(X,y)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[CV] learning_rate=0.1, n_estimators=100 …

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] . learning_rate=0.1, n_estimators=100, score=0.811, total=   2.5s
[CV] learning_rate=0.1, n_estimators=100 …

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.4s remaining:    0.0s

[CV] . learning_rate=0.1, n_estimators=100, score=0.824, total=   2.3s
[CV] learning_rate=0.1, n_estimators=100 …

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    4.7s remaining:    0.0s

[CV] . learning_rate=0.1, n_estimators=100, score=0.823, total=   2.3s
[CV] learning_rate=0.1, n_estimators=200 …
[CV] . learning_rate=0.1, n_estimators=200, score=0.811, total=   4.3s
[CV] learning_rate=0.1, n_estimators=200 …
[CV] . learning_rate=0.1, n_estimators=200, score=0.824, total=   4.4s
[CV] learning_rate=0.1, n_estimators=200 …
[CV] . learning_rate=0.1, n_estimators=200, score=0.823, total=   4.4s
[CV] learning_rate=0.1, n_estimators=400 …
[CV] . learning_rate=0.1, n_estimators=400, score=0.811, total=   8.6s
[CV] learning_rate=0.1, n_estimators=400 …
[CV] . learning_rate=0.1, n_estimators=400, score=0.824, total=   8.5s
[CV] learning_rate=0.1, n_estimators=400 …
[CV] . learning_rate=0.1, n_estimators=400, score=0.823, total=   8.7s
[CV] learning_rate=0.01, n_estimators=100 …
[CV]  learning_rate=0.01, n_estimators=100, score=0.811, total=   2.2s
[CV] learning_rate=0.01, n_estimators=100 …
[CV]  learning_rate=0.01, n_estimators=100, score=0.824, total=   2.2s
[CV] learning_rate=0.01, n_estimators=100 …
[CV]  learning_rate=0.01, n_estimators=100, score=0.823, total=   2.2s
[CV] learning_rate=0.01, n_estimators=200 …
[CV]  learning_rate=0.01, n_estimators=200, score=0.811, total=   4.4s
[CV] learning_rate=0.01, n_estimators=200 …
[CV]  learning_rate=0.01, n_estimators=200, score=0.824, total=   4.4s
[CV] learning_rate=0.01, n_estimators=200 …
[CV]  learning_rate=0.01, n_estimators=200, score=0.823, total=   4.4s
[CV] learning_rate=0.01, n_estimators=400 …
[CV]  learning_rate=0.01, n_estimators=400, score=0.811, total=   8.6s
[CV] learning_rate=0.01, n_estimators=400 …
[CV]  learning_rate=0.01, n_estimators=400, score=0.824, total=   8.7s
[CV] learning_rate=0.01, n_estimators=400 …
[CV]  learning_rate=0.01, n_estimators=400, score=0.823, total=   8.5s
[CV] learning_rate=0.03, n_estimators=100 …
```

```
[CV]  learning_rate=0.03, n_estimators=100, score=0.811, total=   2.5s
[CV] learning_rate=0.03, n_estimators=100 …
[CV]  learning_rate=0.03, n_estimators=100, score=0.824, total=   2.5s
[CV] learning_rate=0.03, n_estimators=100 …
[CV]  learning_rate=0.03, n_estimators=100, score=0.823, total=   2.3s
[CV] learning_rate=0.03, n_estimators=200 …
[CV]  learning_rate=0.03, n_estimators=200, score=0.811, total=   4.5s
[CV] learning_rate=0.03, n_estimators=200 …
[CV]  learning_rate=0.03, n_estimators=200, score=0.824, total=   4.5s
[CV] learning_rate=0.03, n_estimators=200 …
[CV]  learning_rate=0.03, n_estimators=200, score=0.823, total=   4.7s
[CV] learning_rate=0.03, n_estimators=400 …
[CV]  learning_rate=0.03, n_estimators=400, score=0.811, total=   8.8s
[CV] learning_rate=0.03, n_estimators=400 …
[CV]  learning_rate=0.03, n_estimators=400, score=0.824, total=   8.7s
[CV] learning_rate=0.03, n_estimators=400 …
[CV]  learning_rate=0.03, n_estimators=400, score=0.823, total=   9.4s

[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:  2.3min finished
```

```
[24]: GridSearchCV(cv=3, error_score=nan,
             estimator=XGBRFClassifier(base_score=0.5, colsample_bylevel=1,
                                       colsample_bynode=0.8, colsample_bytree=1,
                                       gamma=0, learning_rate=1,
                                       max_delta_step=0, max_depth=3,
                                       min_child_weight=1, missing=None,
                                       n_estimators=100, n_jobs=1, nthread=None,
                                       objective='binary:logistic',
                                       random_state=0, reg_alpha=0,
                                       reg_lambda=1, scale_pos_weight=1,
                                       seed=None, silent=None, subsample=0.8,
                                       verbosity=1),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.1, 0.01, 0.03],
                         'n_estimators': [100, 200, 400]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=3)
```

```
[25]: print(grid.best_params_)
      print(grid.best_score_)
```

```
{'learning_rate': 0.1, 'n_estimators': 400}
0.8194166666666667
```

```
[27]: model_xgb = XGBRFClassifier(n_estimators=400,learning_rate=0.1)
      #model_rf.fit(train_data.
       ↪drop([target,ID,'Balance_Limit_V1','Gender','EDUCATION_STATUS','MARITAL_STATUS','AGE'],axis
      #model_rf.fit(X_train,y_train)
```

```
scores = cross_val_score(model_xgb, X, y, cv=5, scoring='accuracy')
print(scores)
print (np.mean(scores))
```

```
[0.80541667 0.81229167 0.82375    0.83104167 0.821875  ]
0.818875
```

[28]:

```
X_train, X_test, y_train, y_test = train_test_split(train_data.
 ↪drop([target,ID,'Balance_Limit_V1','Gender','EDUCATION_STATUS','MARITAL_STATUS','AGE'],axis
                                                     train_data[target],␣
 ↪test_size=0.30)
model_xgb.fit(X_train,y_train)
```

[28]:

```
XGBRFClassifier(base_score=0.5, colsample_bylevel=1, colsample_bynode=0.8,
                colsample_bytree=1, gamma=0, learning_rate=0.1,
                max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
                n_estimators=400, n_jobs=1, nthread=None,
                objective='binary:logistic', random_state=0, reg_alpha=0,
                reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
                subsample=0.8, verbosity=1)
```

[29]:

```
preds_xgb = model_xgb.predict(X_test)

print(classification_report(y_test,preds_xgb))
print ('\n')
print(confusion_matrix(y_test,preds_xgb))
```

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89      5612
           1       0.69      0.32      0.44      1588

    accuracy                           0.82      7200
   macro avg       0.76      0.64      0.67      7200
weighted avg       0.80      0.82      0.79      7200



[[5385  227]
 [1073  515]]
```

0.0.4 Implementation of the model in test data

[33]:

```
model_xgb = XGBRFClassifier(n_estimators=400,learning_rate = 0.1)
model_xgb.fit(train_data.
 ↪drop([target,ID,'Balance_Limit_V1','Gender','EDUCATION_STATUS','MARITAL_STATUS','AGE'],axis
```

[33]:

```
XGBRFClassifier(base_score=0.5, colsample_bylevel=1, colsample_bynode=0.8,
                colsample_bytree=1, gamma=0, learning_rate=0.1,
                max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
                n_estimators=400, n_jobs=1, nthread=None,
                objective='binary:logistic', random_state=0, reg_alpha=0,
                reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
                subsample=0.8, verbosity=1)
```

[35]:

```
preds_xgb = model_xgb.predict(test_data.
 ↪drop([ID,'Balance_Limit_V1','Gender','EDUCATION_STATUS','MARITAL_STATUS','AGE'],axis=1))
```

[36]:

```
sample_submission = pd.DataFrame(columns=[ID,target])
sample_submission[ID]=test_data[ID]
sample_submission[target] = preds_xgb
```

[37]:

```
submission = sample_submission.to_csv('data-storm-day1-3.csv',index = None)
```