26.Loan_predict_phase3-XGBoost

February 8, 2022

[1]:
```
#import packages to do EDA
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

import os
%matplotlib inline
plt.style.use('fivethirtyeight')


import gc

# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
import matplotlib.font_manager


from sklearn.model_selection import train_test_split
from tqdm import tqdm
import pickle

from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix,roc_curve, auc
from sklearn.metrics import log_loss
import optuna

import warnings
warnings.filterwarnings("ignore")
```

Read vector from pickle file

[2]:
```
with open('./data/train_vector.pkl', 'rb') as f:
    X = pickle.load(f)
```

[3]:
```
with open('./data/feature_names.pkl', 'rb') as f:
    feature_names = pickle.load(f)
```

[4]:
```
print(X.shape[1],len(feature_names))
```

930 930

[5]:
```
with open('./data/yvalues.pkl', 'rb') as f:
    y = pickle.load(f)
```

1 testing purpose only

traindata=5000 X = X.toarray()[:traindata,:] y=y[:traindata] print(X.shape,y.shape)

[6]:
```
#split data only train and test.
#Hypertuning with gridsearch and random
#hypertuning, will do automatic cv. Hence, split data into Train and Test only.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
↪stratify=y,random_state=42)
```

[11]:
```
from xgboost import XGBClassifier
```

[ ]:
```

```

Model Evaluation

Hyper parameter tuning for ensemble models

[7]:
```
#credit:https://www.kaggle.com/saurabhshahane/
↪lgbm-hyperparameter-tuning-with-optuna-beginners
#credit:https://towardsdatascience.com/
↪kagglers-guide-to-lightgbm-hyperparameter-tuning-with-optuna-in-2021-ed048d9838b5
from sklearn.metrics import log_loss
from sklearn.model_selection import StratifiedKFold
from optuna import Trial
from optuna.integration import XGBoostPruningCallback

from scipy.stats import randint as sp_randint
from sklearn.metrics import roc_auc_score

def objective(trial, X_data, y_label):
    param_grid = {
        "verbosity": 0,
        "objective" : trial.suggest_categorical("objective", ['binary:
↪logistic']),
        "n_estimators": trial.suggest_int('n_estimators', 50, 10000),
        "learning_rate": trial.suggest_categorical('learning_rate', [0.008,0.
↪009,0.01,0.012,0.014,0.016,0.018, 0.02]),
        "alpha": trial.suggest_loguniform('alpha', 1e-3, 10.0),
        'gamma': trial.suggest_loguniform('gamma', 1e-3, 10.0),
        "lambda": trial.suggest_loguniform('lambda', 1e-3, 10.0),
```

```
        "min_child_weight": trial.suggest_int('min_child_weight', 1, 200),
        'subsample' : trial.suggest_loguniform('subsample', 0.5, 1),
        'colsample_bytree': trial.suggest_loguniform('colsample_bytree', 0.5,␣
↪1),
        'colsample_bylevel': trial.suggest_loguniform('colsample_bylevel', 0.5,␣
↪1),
        "max_depth": trial.suggest_categorical('max_depth',␣
↪[5,7,9,11,13,15,17,20]),
        }

    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    cv_scores = np.empty(5)

    for idx, (train_idx, valid_idx) in enumerate(cv.split(X_data, y_label)):
        x_train, x_valid = X_data[train_idx], X_data[valid_idx]
        y_train, y_valid = y_label[train_idx], y_label[valid_idx]

        #Define weight ratio
        weight_ratio = float(len(y_train[y_train == 0]))/
↪float(len(y_train[y_train ==1]))
        w_array = np.array([1]*y_train.shape[0])#positive class
        w_array[y_train==1] = weight_ratio
        w_array[y_train==0] = 1- weight_ratio #negative class

        #XGBoostClassifier with random_state=0
        #scale_pos_weight - A value greater than 0 should be used
        #in case of high class imbalance as it helps in faster convergence

        model = XGBClassifier(tree_method = "exact",predictor = "cpu_predictor",
                                                 ␣
↪scale_pos_weight=(weight_ratio*100),random_state=0)
        model.set_params(**param_grid)

        model.fit(x_train,y_train, eval_set=[(x_valid, y_valid)], verbose=0,
                         ␣
↪early_stopping_rounds=50,callbacks=[XGBoostPruningCallback(trial,␣
↪'validation_0-logloss')]
                  )

        preds = model.predict_proba(x_valid)
        cv_scores[idx] = roc_auc_score(y_valid, preds[:,1])

    return np.mean(cv_scores)
```

[8]:
```
from warnings import simplefilter
simplefilter("ignore", category=RuntimeWarning)
```

```
optuna.logging.set_verbosity(optuna.logging.WARNING)
study = optuna.create_study(direction="minimize", study_name="XGBoost␣
↪Classifier" )
func = lambda trial: objective(trial, X_train, y_train)
study.optimize(func, n_trials=100)
```

[9]:
```
print(f"\tBest value (rmse): {study.best_value:.5f}")
print(f"\tBest params:")

for key, value in study.best_params.items():
    print(f"\t\t{key}: {value}")
```

```
        Best value (rmse): 0.66379
        Best params:
                objective: binary:logistic
                n_estimators: 5509
                learning_rate: 0.008
                alpha: 0.0014764404145375422
                gamma: 0.006709171934295603
                lambda: 0.004841559984476445
                min_child_weight: 18
                subsample: 0.801485530162779
                colsample_bytree: 0.8775747304615185
                colsample_bylevel: 0.5456219632677068
                max_depth: 20
```

[42]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
↪estimates of the positive class
    # not the predicted outputs
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 -␣
↪49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
    return y_data_pred
```

[43]:
```
#https://stackoverflow.com/questions/61748441/
↪how-to-fix-the-values-displayed-in-a-confusion-matrix-in-exponential-form-to-nor
def plot_confusionmatrix(y_tr,y_trpred,y_te,y_tepred):
    from sklearn.metrics import confusion_matrix
```

```
    tn, fp, fn, tp = confusion_matrix(y_tr, np.round(y_trpred)).ravel()
    print('Traiing data tn-> {}, fp-> {}, fn-> {}, tp-> {}'.format(tn, fp, fn,␣
↪tp), end=" ")
    #confusion matrix on training data
    plt.figure(figsize=(10, 10))
    ax_tr = plt.subplot(221)
    cm_tr = confusion_matrix(y_tr, np.round(y_trpred))
    plt.title("Training data - Confusion Matrix")
    sns.heatmap(cm_tr, ax=ax_tr, fmt='d',cmap='YlGnBu',annot=True)
    # labels, title and ticks
    ax_tr.set_xlabel('Predicted labels');
    ax_tr.set_ylabel('True labels');
    ax_tr.set_ylim(2.0, 0)
    ax_tr.xaxis.set_ticklabels(['No','Yes']);
    ax_tr.yaxis.set_ticklabels(['No','Yes']);

    #Confusion matrix on test data
    tn, fp, fn, tp = confusion_matrix(y_te, np.round(y_tepred)).ravel()
    print('Traiing data tn-> {}, fp-> {}, fn-> {}, tp-> {}'.format(tn, fp, fn,␣
↪tp), end=" ")

    ax_te = plt.subplot(222)
    cm_te = confusion_matrix(y_te, np.round(y_tepred))
    plt.title("Test data - Confusion Matrix")
    sns.heatmap(cm_te, ax=ax_te, fmt='d',cmap='YlGnBu',annot=True)
    # labels, title and ticks
    ax_te.set_xlabel('Predicted labels');
    ax_te.set_ylabel('True labels');
    ax_te.set_ylim(2.0, 0)
    ax_te.xaxis.set_ticklabels(['No','Yes']);
    ax_te.yaxis.set_ticklabels(['No','Yes']);

    plt.show()
    return
```

[44]:
```
def draw_roccurve(y_tr,y_tr_pred,y_te,y_te_pred):
    #fpr,tpr,thresholds
    fpr, tpr, thresholds = roc_curve(y_tr, np.array(y_tr_pred))
    #auc score train score
    auc_train = round(auc(fpr, tpr),5)
    plt.plot(fpr, tpr, label=" AUC train ="+str(auc_train))
    plt.plot([0, 1], [0, 1],'r--')

    fpr, tpr, thresholds = roc_curve(y_te, np.array(y_te_pred))
    #auc score test score
    auc_test = round(auc(fpr, tpr),5)
    plt.plot(fpr, tpr, label=" AUC test ="+str(auc_test))
```

```
    plt.plot([0, 1], [0, 1],'b--')

    plt.legend()
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("ROC" )
    plt.grid()
    plt.show()
    return auc_train,auc_test
```

[52]:
```
from sklearn.utils import class_weight
#class_weights = class_weight.compute_class_weight('balanced', np.
 ↪unique(y_train),y_train)
#class_weights = dict(zip(np.unique(y_train), class_weight.
 ↪compute_class_weight('balanced', np.unique(y_train),y_train)))
#y_integers = np.argmax(y_train, axis=1)
class_weights = class_weight.compute_class_weight(class_weight='balanced',␣
 ↪classes=[0,1], y=y_train)
d_class_weights = dict(enumerate(class_weights))
d_class_weights
```

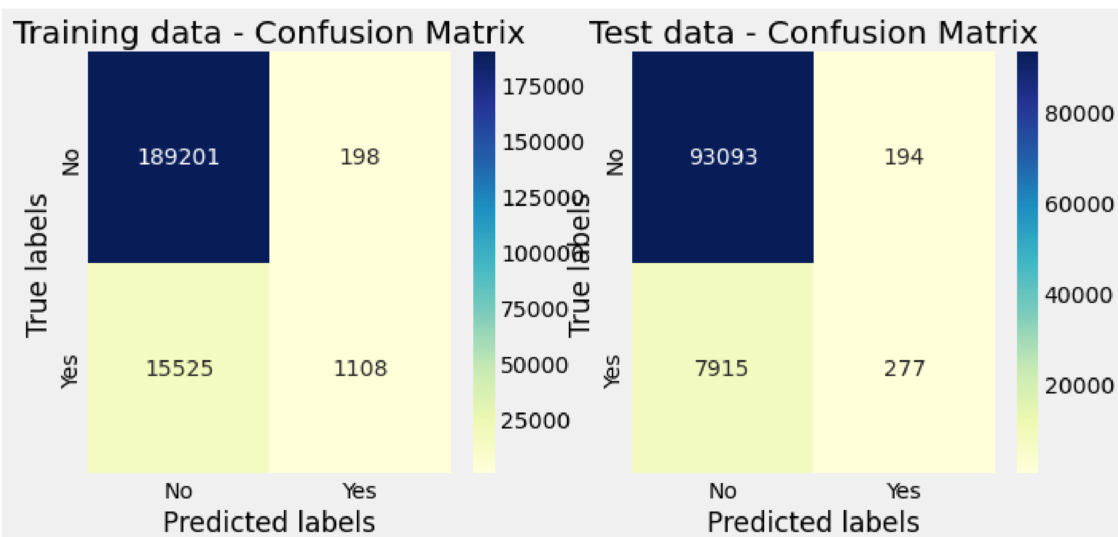
[52]: {0: 0.5439099467262235, 1: 6.193470811038297}

[53]:
```
#based on the best parameters, predict values and plot AUC  and return the model
#def measure_accuracy(study,X_tr,X_te,y_tr,y_te):
def measure_accuracy(X_tr,X_te,y_tr,y_te):

    #clf = XGBClassifier(**study.best_params) # overfitting with optuna results.
 ↪ Hence manually changed the params as below.
    param_grid = {
        "verbosity": 0,
        "objective" :'binary:logistic',
        "n_estimators":10000,
        "learning_rate": 0.001,
        "alpha": 0.001,
        'gamma': 0.005,
        "lambda": 0.004,
        "min_child_weight": 20,
        'subsample' : 0.8,
        'colsample_bytree': 0.9,
        'colsample_bylevel': 0.5,
        "max_depth": 7,
        }

    clf = XGBClassifier(tree_method = "exact",predictor =␣
 ↪"cpu_predictor",sample_weight=d_class_weights,random_state=0)
    clf.set_params(**param_grid)
```

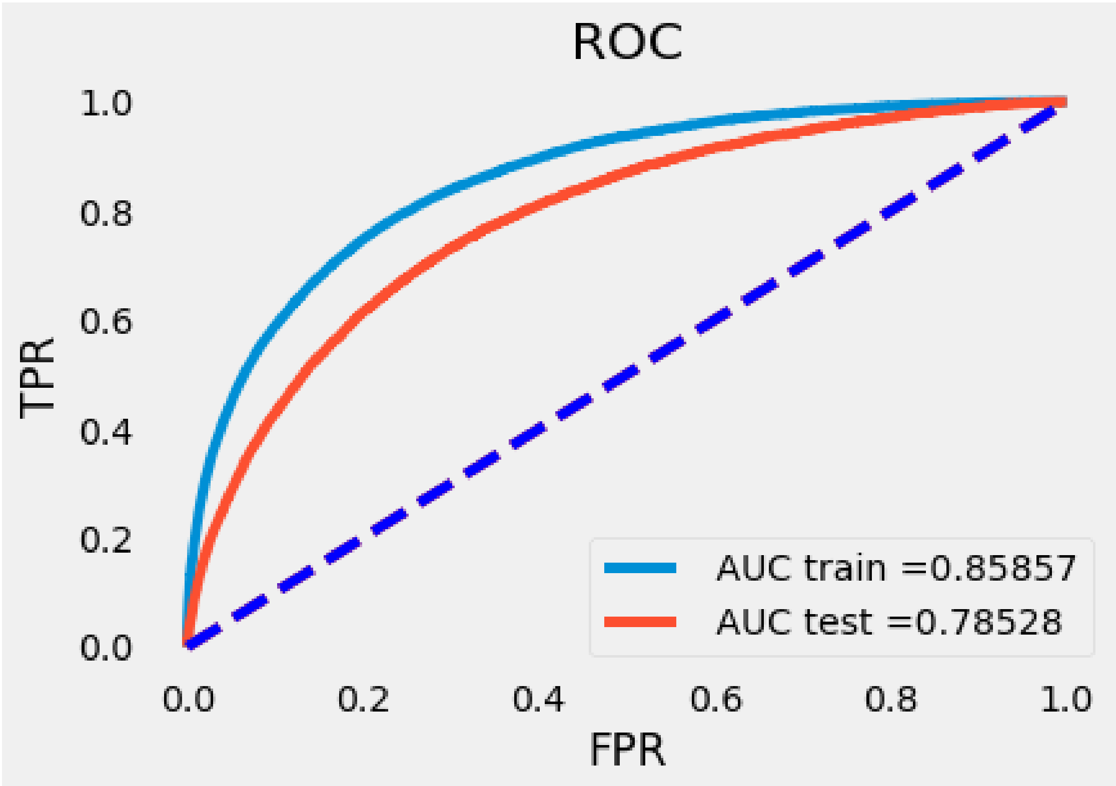
```
    clf.fit(X_tr,y_tr,verbose=False, )

    y_tr_pred = batch_predict(clf, X_tr)
    y_te_pred = batch_predict(clf, X_te)

    plot_confusionmatrix(y_tr,y_tr_pred,y_te,y_te_pred)
    print('='*70)
    auc_train,auc_test=draw_roccurve(y_tr,y_tr_pred,y_te,y_te_pred)
    print('='*70)
    return clf, auc_train,auc_test
```

[54]:
```
#xgb_model,auc_tr_xgb_model,auc_te_xgb_model =␣
 ↪measure_accuracy(study,X_train,X_test,y_train,y_test)
xgb_model,auc_tr_xgb_model,auc_te_xgb_model =␣
 ↪measure_accuracy(X_train,X_test,y_train,y_test)
```

```
Traiing data tn-> 189201, fp-> 198, fn-> 15525, tp-> 1108 Traiing data tn->
93093, fp-> 194, fn-> 7915, tp-> 277
```

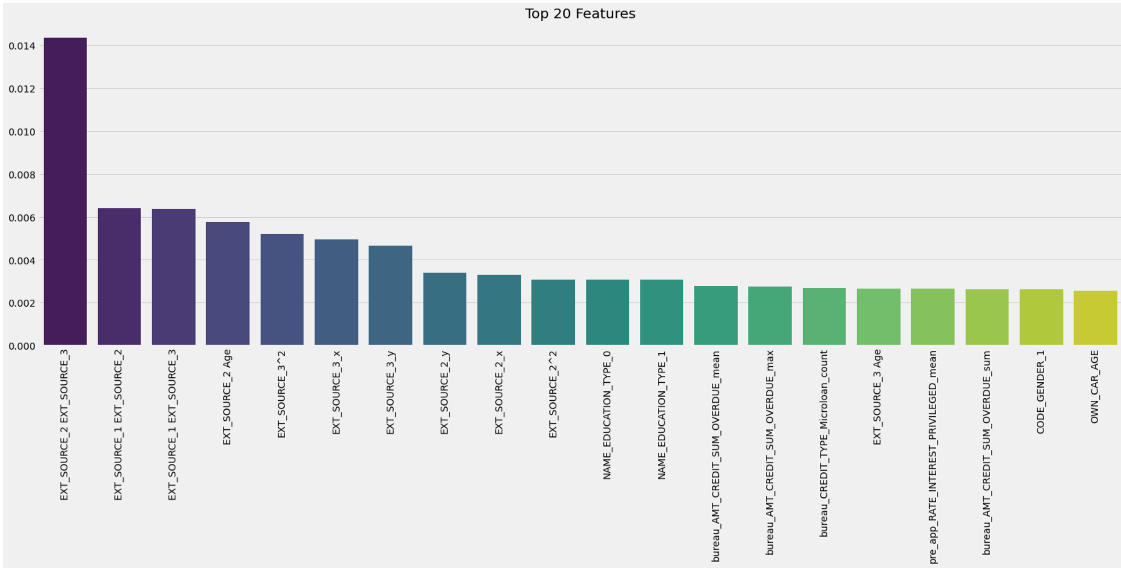



```
======================================================================
```



==

[55]:
```
top20_feature_names=[]
feature_importance = xgb_model.feature_importances_
feature_importances = (xgb_model.feature_importances_ / sum(xgb_model.
 ↪feature_importances_+0.000001)) * 100
indices = feature_importance.argsort()[::-1][:20]
for i in indices:
    top20_feature_names.append(feature_names[i])

#Plot bar plot for top 15 features
plt.close()
column =top20_feature_names
score = feature_importance[indices]
plt.figure(figsize =(25, 8))
sns.barplot(x=column, y=score, palette="viridis")
plt.xticks(rotation=90)
plt.title('Top 20 Features')
plt.show()
```



[56]:
```python
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Feature","Score"]
for val in zip(column,score):
    x.add_row([val[0],np.round(val[1],4)])

x.sortby = "Score"
x.reversesort = True
x.align["Feature"] = "l"
x.align["Score"] = "r"

print(x)
```

```
+----------------------------------------+--------+
| Feature                                |  Score |
+----------------------------------------+--------+
| EXT_SOURCE_2 EXT_SOURCE_3              | 0.0144 |
| EXT_SOURCE_1 EXT_SOURCE_3              | 0.0064 |
| EXT_SOURCE_1 EXT_SOURCE_2              | 0.0064 |
| EXT_SOURCE_2 Age                       | 0.0058 |
| EXT_SOURCE_3^2                         | 0.0052 |
| EXT_SOURCE_3_x                         |  0.005 |
| EXT_SOURCE_3_y                         | 0.0047 |
| EXT_SOURCE_2_y                         | 0.0034 |
| EXT_SOURCE_2_x                         | 0.0033 |
| NAME_EDUCATION_TYPE_1                  | 0.0031 |
| NAME_EDUCATION_TYPE_0                  | 0.0031 |
```

```
| EXT_SOURCE_2^2                         | 0.0031 |
| bureau_AMT_CREDIT_SUM_OVERDUE_mean     | 0.0028 |
| bureau_AMT_CREDIT_SUM_OVERDUE_max      | 0.0028 |
| bureau_CREDIT_TYPE_Microloan_count     | 0.0027 |
| EXT_SOURCE_3 Age                       | 0.0027 |
| pre_app_RATE_INTEREST_PRIVILEGED_mean  | 0.0026 |
| bureau_AMT_CREDIT_SUM_OVERDUE_sum      | 0.0026 |
| OWN_CAR_AGE                            | 0.0026 |
| CODE_GENDER_1                          | 0.0026 |
+----------------------------------------+--------+
```

[57]:
```
import joblib
# save model
joblib.dump(xgb_model, './results/model_xgboost.pkl')
```

[57]: ['./results/model_xgboost.pkl']