Financial Risk Control

In this project, I would utilize the AI tools to determine whether to give a person loan or not. Deciding whether to provide loans for a person would take as long as a few weeks, but AI could process all the information and make reasonable prediction for thousands of cliends in less than 10 seconds.

2. Data Exploration

In this project, I would use the training set: `Training/PPD_Training_Master_GBK_3_1_Training_Set.csv` to train my model. Then, testing this model for accuracy on the testing set: `Test/PPD_Master_GBK_2_Test_Set.csv` .

The data includes clients' `living places` , `education level` , `weblog Inforation` and in total of 227 independent x variables and the y variable is labeled in 1 or 0. 1 represents giving loans and 0 represents not giving loans.

3. Abstract

---

1. Clean the data: clean the variables which have missing values, combine string, and discard some irrelevant factors.

---

1. Choose variables: I expand some discrete variables into columns by one-hot encoding method. I use `Decision Tree` algorisms to select best variables.

---

1. Create model:I also use gridsearch to select variables and input the variables selected into the `XGBoost` model which is a typical nonlinear boosting method to train the data and eventually test it on the testing set.

4. Detailed Code

In [37]:

```
import numpy as np
import math
import pandas as pd
pd.set_option('display.float_format',lambda x:'%.3f' % x)
import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline
import seaborn as sns
sns.set_palette('muted')
sns.set_style('darkgrid')
import warnings
```

```
warnings.filterwarnings('ignore')
import os
```

In [38]:

```
# read the data
train= pd.read_csv('train/PPD_Training_Master_GBK_3_1_Training_Set.csv',encoding
test= pd.read_csv('test/PPD_Master_GBK_2_Test_Set.csv',encoding='gb18030')
data=pd.concat([train,test],sort=False)
```

In [39]:

```
train_size=len(train)
test_size=len(test)
data.head()
```

Out[39]:

	Idx	UserInfo_1	UserInfo_2	UserInfo_3	UserInfo_4	WeblogInfo_1	WeblogInfo_2	WeblogInf
0	10001	1.000	深圳	4.000	深圳	NaN	1.000	
1	10002	1.000	温州	4.000	温州	NaN	0.000	
2	10003	1.000	宜昌	3.000	宜昌	NaN	0.000	
3	10006	4.000	南平	1.000	南平	NaN	NaN	
4	10007	5.000	辽阳	1.000	辽阳	NaN	0.000	

5 rows × 228 columns

In [40]:

```
# check for number of null values in variables
data.target.value_counts()
null_value=[]
def statics(data):
    for col in data.columns:
        null_value.append([col,data[col].isnull().sum()/len(data),data[col].dtyp
statics(data)
null_value=pd.DataFrame(null_value,columns=['column_name','percentage_of_null_va
null_value=null_value.sort_values(by='percentage_of_null_value',ascending=False)
null_value.head(20)
```

Out[40]:

	column_name	percentage_of_null_value	Data_type	Unique
7	WeblogInfo_3	0.968	float64	24
5	WeblogInfo_1	0.968	float64	25
30	UserInfo_12	0.632	float64	3
31	UserInfo_13	0.632	float64	3
29	UserInfo_11	0.632	float64	3
226	target	0.400	float64	3
52	WeblogInfo_20	0.267	object	39
53	WeblogInfo_21	0.101	object	5
51	WeblogInfo_19	0.098	object	8
6	WeblogInfo_2	0.056	float64	6
9	WeblogInfo_5	0.056	float64	40

	column_name	percentage_of_null_value	Data_type	Unique
10	WeblogInfo_6	0.056	float64	65
8	WeblogInfo_4	0.056	float64	67
2	UserInfo_2	0.009	object	330
63	WeblogInfo_32	0.009	float64	5
69	WeblogInfo_38	0.009	float64	5
68	WeblogInfo_37	0.009	float64	6
66	WeblogInfo_35	0.009	float64	4
65	WeblogInfo_34	0.009	float64	6
64	WeblogInfo_33	0.009	float64	20

In [41]:

```
# drop columns with large null percentage
large_null_columns=list(null_value.iloc[0:5,0])
data.drop(columns=large_null_columns,inplace=True)
data.drop(columns='ListingInfo',inplace=True)
```

In [42]:

```
#fill the null value
data.fillna(-199,inplace=True)
data["UserInfo_2"]=data['UserInfo_2'].astype(str)
data['UserInfo_4']=data['UserInfo_4'].astype(str)

data.UserInfo_9=data.UserInfo_9.str.replace(" ",'')

data.UserInfo_19=data.UserInfo_19.str.replace('省','')
data.UserInfo_19=data.UserInfo_19.str.replace('市','')
data.UserInfo_20=data.UserInfo_20.str.replace('省','')
data.UserInfo_20=data.UserInfo_20.str.replace('市','')
```

In [43]:

```
## delete all variable with values over 3000 and get dummies to values whose uni
unique_value_lessthan10=[]
unique_value_over3000=[]
for col in data.columns:
    if col not in ['Idx','target']:
        if data[col].nunique()<=10:
            unique_value_lessthan10.append(col)
        elif data[col].nunique()>=1000:
            unique_value_over3000.append(col)
unique_value_over3000

data.drop(columns=unique_value_over3000,inplace=True)
data=pd.get_dummies(data,columns=unique_value_lessthan10)
```

In [44]:

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
object_column=list(data.columns[data.dtypes=='object'])
for i in object_column:
   data[i]=le.fit_transform(data[i].astype(str))
```

In [45]:
```
## Split the data into training and testing set.
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE

data_train=data.iloc[:train_size,data.columns!='target'].values
data_test=data.target[:train_size].values
X_train,x_test,y_train,y_test=train_test_split(data_train,data_test,test_size=0.
#data_test_unknown_target=data.iloc[train_size:,data.columns!='target']
```

In [46]:
```

#create tree model
dt_clf = DecisionTreeClassifier(class_weight='balanced')

#create grid for parameters
params_min_samples_split = [5,10,15,20]

params_min_samples_leaf = [2,4,6,8,10]

params_max_depth = [4,6,8,10]

param_grid_dt = {'min_samples_split' : params_min_samples_split,
                 'min_samples_leaf' : params_min_samples_leaf,
                 'max_depth' : params_max_depth}
#use GridSearchCV method and using five-fold training, the scoring method is auc
grid_dt = GridSearchCV(estimator=dt_clf, param_grid=param_grid_dt, cv=5, scoring
#fit into the dataset
grid_dt.fit(X_train, y_train)
#use grid.best_params_ to get the best parameters
print('best parameter:{}'.format(grid_dt.best_params_))
print('best score:{}'.format(grid_dt.best_score_))
print('best model:{}'.format(grid_dt.best_estimator_))
```

```
best parameter:{'max_depth': 4, 'min_samples_leaf': 8, 'min_samples_split': 15}
best score:0.6348513731892955
best model:DecisionTreeClassifier(class_weight='balanced', max_depth=4, min_samp
les_leaf=8,
                       min_samples_split=15)
```

In [47]:
```

#create best model
model = grid_dt.best_estimator_
#train the model using chosen parameter
model.fit(X_train,y_train)
# select variables
select_model = SelectFromModel(model, prefit=True)
selected_features = select_model.get_support()
# put selected variables into X_train1 and X_test1
X_train1 = X_train[:, selected_features]
X_test1 = x_test[:, selected_features]
# print the shape
print(X_train1.shape, X_test1.shape)
```

```
(24000, 15) (6000, 15)
```

In [48]:
```
#create xgboost model, booster is gbtree, since y variable 0:1 is close to 15:1,
xgb_clf = XGBClassifier(booster='gbtree', scale_pos_weight=15, eval_metric='auc'
#parameters into grid
params_max_depth = [4,6,8,10]

params_n_estimators = [100,200,300,400,500]

params_colsample_bytree = [0.3,0.5,0.7,0.9]

params_subsample = [0.3,0.5,0.7,0.9]
#create gridsearch
param_grid_xgb = {'max_depth' : params_max_depth,
                  'n_estimators' : params_n_estimators,
                  'colsample_bytree' : params_colsample_bytree,
                  'subsample' : params_subsample}
#GridSearchCV five-folded roc_auc
grid_xgb = GridSearchCV(estimator=xgb_clf, param_grid=param_grid_xgb, cv=5, scor
#fit into the data
grid_xgb.fit(X_train1,y_train)
#best parameter
print('best parameter:{}'.format(grid_xgb.best_params_))
#best score
print('best score:{}'.format(grid_xgb.best_score_))
#best model
print('best model:{}'.format(grid_xgb.best_estimator_))
```

```
best parameter:{'colsample_bytree': 0.3, 'max_depth': 4, 'n_estimators': 100, 's
ubsample': 0.9}
best score:0.668010725898491
best model:XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, eval_metric='auc',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=4, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=15, subsample=0.9, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [49]:
```

#create best model
model_xgb = grid_xgb.best_estimator_
#put model into testing set
predictions_xgb = model_xgb.predict(X_test1)
# get auc result
rf_roc_auc = roc_auc_score(y_test, predictions_xgb)
# print the result
print ("AUC = %2.2f" % rf_roc_auc)
print('"\n\n ---Testing result---"')
print(classification_report(y_test, predictions_xgb))
```

```
AUC = 0.65
"


 ---Testing result---"
              precision    recall  f1-score   support

         0.0       0.96      0.66      0.78      5560
         1.0       0.13      0.63      0.21       440

    accuracy                           0.66      6000
   macro avg       0.54      0.65      0.50      6000
```

```
weighted avg       0.90      0.66      0.74      6000
```