Wide and deep neural network

Hyundai Card Internship Program
2018 Summer

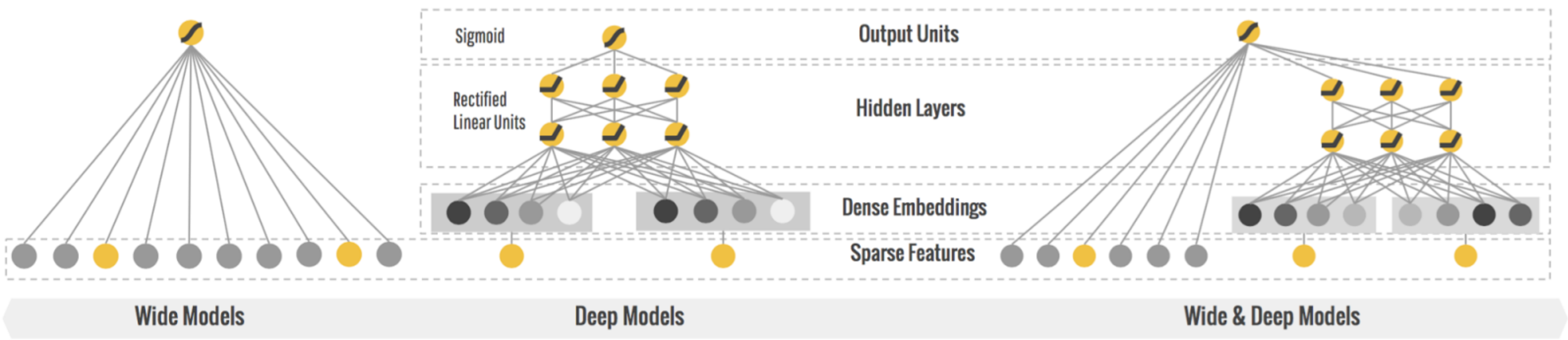
Siqi Shao

General Views

For the financial data, there usually contains comprehensive information, including shallow features and deep features. Effectively put different types of features into right model could boost the value of the data itself. One of the great things in TensorFlow is that it offers different feature columns API give method to convert data between sparse and continuous representation.

Modeling

Wide and deep Neural Network Structure:



Picture original from Wide & Deep Learning for Recommender Systems

- Using high level API from Tensorflow and joint train a linear model and a deep NN together.
=> tf.estimator.DNNLinearCombinedClassifier()

- Combine strengths of memorization and generalization.

- Useful for generic large-scale regression and classification problems with sparse input features.

Data

• Kaggle competition: Home Credit Default Risk

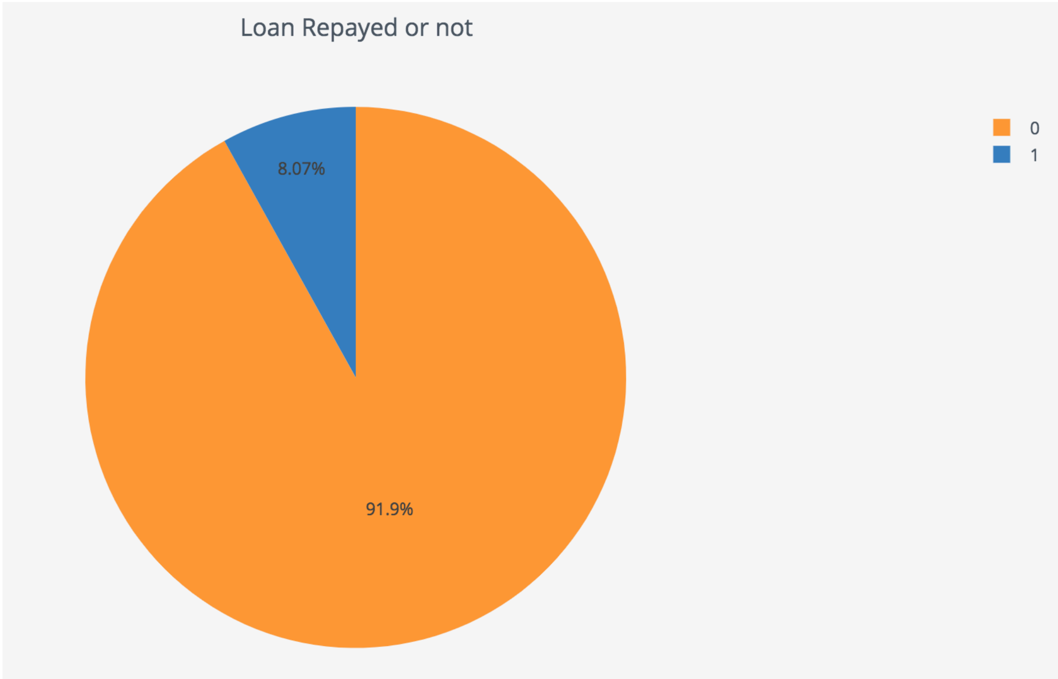
Home Credit Group is a credit card company and predicting whether or not a client will repay a loan or have difficulty is a critical business need for them.

• Dataset overview

- Labeled data for training set: The labels are included in the training dataset and the goal is to train a model and predict the labels.
- There are 7 different sources of data in total. Every loan has its own row and is identified by the feature SK_ID_CURR.
- For this project: 307,511 objects & 122 features (16 Categorical features and 106 numerical features)

```
In [7]: df_train.dtypes.value_counts()

Out[7]: float64    65
        int64      41
        object     16
        dtype: int64
```

- Classification: The label is a binary variable 0 (91.9% will repay loan on time), 1 (8.07% will have difficulty repaying loan).



steps

missing values

dataset is imbalanced or not

the distribution of some feature

feature importance ranking

Exploratory data analysis(EDA)

correlation calculation

Filling missing value

Preprocessing

Scaling

Build feature columns

Remove some features

Feature Selection

Wide and deep columns separation

Modeling

Evaluation

Modularized code and prepared for Airflow

Environment Setup

1. Recommandation environment: Anaconda virtual environment (python 3.6).
2. Install tensorflow 1.8.0 (newest version is 1.9.0).
3. Install core library for scientific computing in Python: numpy, pandas, sklearn, matplotlib,etc.
4. Install feature_selector for https://github.com/WillKoehrsen/feature-selector. (put it in current directory).
5. Dataset from Kaggle competition https://www.kaggle.com/c/home-credit-default-risk/data.

Feature selection

- ## FeatureSelector

```
fs = FeatureSelector(data = train, labels = train_labels)
fs.identify_all(selection_params = {'missing_threshold': 0.6, 'correlation_threshold': 0.95,
                                    'task': 'classification', 'eval_metric': 'auc',
                                     'cumulative_importance': 0.99})
```

- Missing value: Find any columns with a missing fraction greater than a specified threshold.
- Unique value: Find any features that have only a single unique value.
- Highly correlated features: This method finds pairs of collinear features based on the Pearson correlation coefficient.
- Features importance : This method relies on a machine learning model to identify features to remove. It therefore requires a supervised learning problem with labels. The method works by finding feature importances using a gradient boosting machine implemented in the LightGBM library.

- ## Remove features

```
train_removed_all_once = fs.remove(methods = 'all', keep_one_hot = True)
```

```
['missing', 'single_unique', 'collinear', 'zero_importance', 'low_importance'] methods have been r
un
```

Construction feature columns

- Problems: Manually define feature columns only suitable for dataset with a small quantity of features and it is hard to generalize.

```python
def build_model_columns():
  """Builds a set of wide and deep feature columns."""
  # Continuous columns
  EXT_SOURCE_2 = tf.feature_column.numeric_column('EXT_SOURCE_2')
  EXT_SOURCE_1 = tf.feature_column.numeric_column('EXT_SOURCE_1')
  EXT_SOURCE_3 = tf.feature_column.numeric_column('EXT_SOURCE_3')
  AMT_CREDIT = tf.feature_column.numeric_column('AMT_CREDIT')
  AMT_ANNUITY = tf.feature_column.numeric_column('AMT_ANNUITY')

  NAME_CONTRACT_TYPE = tf.feature_column.categorical_column_with_vocabulary_list(
      'NAME_CONTRACT_TYPE', [
          'Cash-loans', 'Revolving-loans'])

  OCCUPATION_TYPE= tf.feature_column.categorical_column_with_vocabulary_list(
      'OCCUPATION_TYPE', [
          'Laborers', 'Core-staff', 'Accountants', 'Managers', 'Drivers',
       'Sales-staff', 'Cleaning-staff', 'Cooking-staff',
       'Private-service-staff', 'Medicine-staff', 'Security-staff',
       'High-skill-tech-staff', 'Waiters-barmen-staff',
       'Low-skill-Laborers', 'Realty-agents', 'Secretaries', 'IT-staff',
       'HR-staff'])

  NAME_INCOME_TYPE= tf.feature_column.categorical_column_with_vocabulary_list(
      'NAME_INCOME_TYPE', [
         'Working', 'State-servant', 'Commercial-associate', 'Pensioner',
       'Unemployed', 'Student', 'Businessman', 'Maternity-leave'])

  NAME_FAMILY_STATUS = tf.feature_column.categorical_column_with_vocabulary_list(
      'NAME_FAMILY_STATUS', [
         'Single-not-married', 'Married', 'Civil-marriage', 'Widow',
       'Separated', 'Unknown'])

  # To show an example of hashing:
  NAME_HOUSING_TYPE = tf.feature_column.categorical_column_with_hash_bucket(
      'NAME_HOUSING_TYPE', hash_bucket_size=1000)

  # Transformations.
  EXT_SOURCE_3_buckets = tf.feature_column.bucketized_column(
      EXT_SOURCE_3, boundaries=[0,0.2,0.4,0.6,0.8,1.0])

  # Wide columns and deep columns.
  base_columns = [
      NAME_CONTRACT_TYPE , OCCUPATION_TYPE, NAME_INCOME_TYPE, NAME_FAMILY_STATUS,NAME_HOUSING_TYPE,
      EXT SOURCE 3 buckets .

 crossed_columns = [
     tf.feature_column.crossed_column(
         ['NAME_CONTRACT_TYPE', 'NAME_HOUSING_TYPE'], hash_bucket_size=1000),
     tf.feature_column.crossed_column(
         [EXT_SOURCE_3_buckets, 'NAME_CONTRACT_TYPE', 'NAME_HOUSING_TYPE'], hash_bucket_size=1000),
 ]

 wide_columns = base_columns + crossed_columns

 deep_columns = [EXT_SOURCE_2,EXT_SOURCE_1,EXT_SOURCE_3,AMT_CREDIT,AMT_ANNUITY,
     tf.feature_column.indicator_column(NAME_CONTRACT_TYPE),
     tf.feature_column.indicator_column(OCCUPATION_TYPE),
     tf.feature_column.indicator_column(NAME_INCOME_TYPE),
     tf.feature_column.indicator_column( NAME_FAMILY_STATUS),
     # To show an example of embedding
     tf.feature_column.embedding_column(NAME_HOUSING_TYPE, dimension=8),

 ]
```

- Solutions:

convert raw data to feature columns — tf.feature_column API

missing value<60%
feature importance top 50
correlation between 0.3-0.4

FeatureSelector

crossed columns

wide categorical + bucketed columns

deep numerical + embedding columns

feed the model & joint training

Correlated features

3	DAYS_BIRTH	CNT_CHILDREN	0.330938
5	DAYS_REGISTRATION	DAYS_BIRTH	0.331912
19	YEARS_BUILD_AVG	APARTMENTS_AVG	0.340784
29	ENTRANCES_AVG	COMMONAREA_AVG	0.326264
43	LANDAREA_AVG	ELEVATORS_AVG	0.375280
47	LIVINGAPARTMENTS_AVG	YEARS_BUILD_AVG	0.333937
56	LIVINGAREA_AVG	YEARS_BUILD_AVG	0.355666
64	NONLIVINGAREA_AVG	LIVINGAREA_AVG	0.302101
67	APARTMENTS_MODE	YEARS_BUILD_AVG	0.323250
78	BASEMENTAREA_MODE	COMMONAREA_AVG	0.388210
99	COMMONAREA_MODE	ENTRANCES_AVG	0.333582
100	COMMONAREA_MODE	FLOORSMAX_AVG	0.378213
160	LANDAREA_MODE	ELEVATORS_AVG	0.361733
167	LANDAREA_MODE	ELEVATORS_MODE	0.379209
171	LIVINGAPARTMENTS_MODE	YEARS_BUILD_AVG	0.326195
182	LIVINGAPARTMENTS_MODE	YEARS_BUILD_MODE	0.332568
191	LIVINGAREA_MODE	YEARS_BUILD_AVG	0.338392
202	LIVINGAREA_MODE	YEARS_BUILD_MODE	0.345003
467	TOTALAREA_MODE	YEARS_BUILD_AVG	0.359263
476	TOTALAREA_MODE	NONLIVINGAREA_AVG	0.366604
479	TOTALAREA_MODE	YEARS_BUILD_MODE	0.355718
488	TOTALAREA_MODE	NONLIVINGAREA_MODE	0.346957
491	TOTALAREA_MODE	YEARS_BUILD_MEDI	0.357972
500	TOTALAREA_MODE	NONLIVINGAREA_MEDI	0.361627
501	DEF_30_CNT_SOCIAL_CIRCLE	OBS_30_CNT_SOCIAL_CIRCLE	0.329338
503	OBS_60_CNT_SOCIAL_CIRCLE	DEF_30_CNT_SOCIAL_CIRCLE	0.331571

Create feature columns

```
YEARS_BUILD_AVG_c=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('YEARS_BUILD_AVG'),
                                                       boundaries = [0,0.2,0.4,0.6,0.8])
APARTMENTS_AVG_c=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('APARTMENTS_AVG'),
                                                       boundaries = [0,0.2,0.4,0.6,0.8])
crossed_col_5 = tf.feature_column.crossed_column( [YEARS_BUILD_AVG_c, APARTMENTS_AVG_c], 5000)
```

```
DAYS_BIRTH=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('DAYS_BIRTH'),
                                                       boundaries = [-25000,-20000,-15000,-10000])
CNT_CHILDREN=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('CNT_CHILDREN'),
                                                       boundaries = [2,4,6,8,10])
crossed_col_1 = tf.feature_column.crossed_column( [DAYS_BIRTH, CNT_CHILDREN], 5000)
```

```
DEF_30_CNT_SOCIAL_CIRCLE=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('DEF_30_CNT_SOCIAL_
                                                                     boundaries = [2,4,6,8])
OBS_30_CNT_SOCIAL_CIRCLE=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('OBS_30_CNT_SOCIAL_
                                                       boundaries = [5, 10,15,20,25,30])
crossed_col_2 = tf.feature_column.crossed_column([DEF_30_CNT_SOCIAL_CIRCLE,OBS_30_CNT_SOCIAL_CIRCLE], 5000)
```

```
DEF_30_CNT_SOCIAL_CIRCLE=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('DEF_30_CNT_SOCIAL_
                                                                    boundaries = [2,4,6,8])
OBS_60_CNT_SOCIAL_CIRCLE=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('OBS_60_CNT_SOCIAL_
                                                       boundaries = [5, 10,15,20,25,30])
crossed_col_3 = tf.feature_column.crossed_column([DEF_30_CNT_SOCIAL_CIRCLE,OBS_30_CNT_SOCIAL_CIRCLE], 5000)
```

```
DAYS_REGISTRATION=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('DAYS_REGISTRATION'),
                                                       boundaries = [-25000,-20000,-15000,-10000,-5000,0])
DAYS_BIRTH=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('DAYS_BIRTH'),
                                                       boundaries = [-25000,-20000,-15000,-10000])
crossed_col_4 = tf.feature_column.crossed_column( [DAYS_BIRTH, DAYS_REGISTRATION], 5000)
```

Result: Feature columns split function can be generalized. Features with correlation between 0.3 to 0.4 could create a good crossed feature.

Modeling & Evaluation

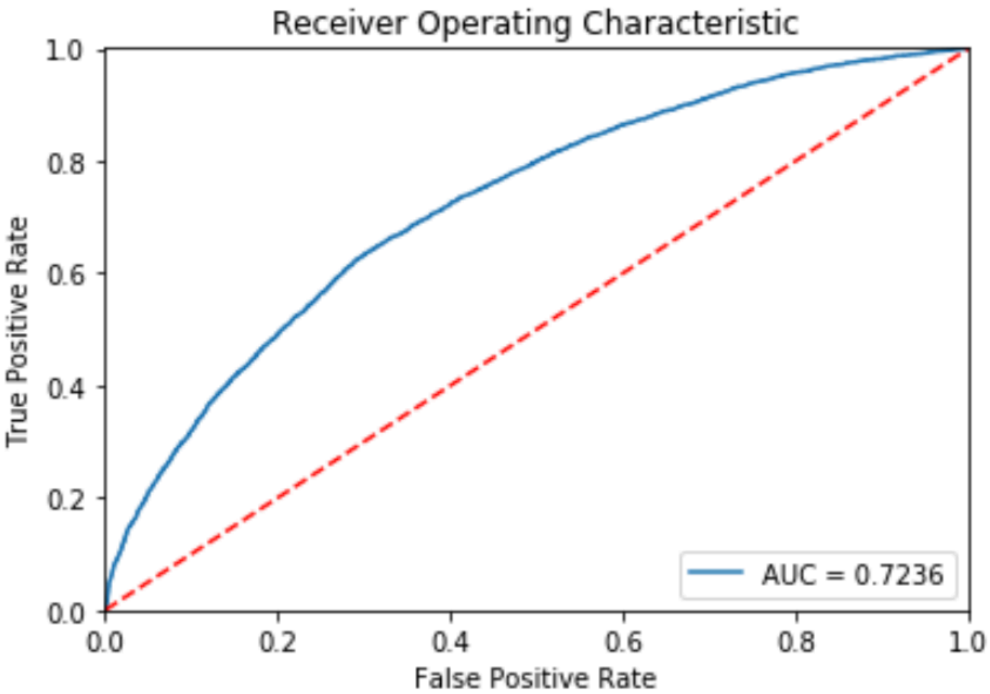
- wide part = sparse columns + crossed columns
- deep part = continuous columns + embedded categorical column
- Using hashing and bucketizing to transform between categorical and numerical data.
- Joint train Wide & Deep model (`DNNLinearCombinedClassifier`).

```
estimator = tf.estimator.DNNLinearCombinedClassifier(linear_feature_columns=wide_columns, dnn_feature_columns=deep_colu
                                                      dnn_hidden_units=[500,150,50], dnn_activation_fn=tf.nn.relu,
                                                      dnn_dropout=0.5,config=run_config)
```

Key for improving the evaluation result :

- Normalizing and scaling data
- Undersampling
- Adam Optimization Algorithm for Deep Learning
- Batch size
- Overfitting: drop out

Result:




```
------------------------------------------------------------------------------------------------------------
{'accuracy': 0.6637396, 'accuracy_baseline': 0.50237656, 'auc': 0.72367144, 'auc_precision_recall': 0.71066856, 'aver
age_loss': 0.61779857, 'label/mean': 0.49762347, 'loss': 7668.734, 'precision': 0.6574438, 'prediction/mean': 0.50592
36, 'recall': 0.6770277, 'global_step': 10000}
------------------------------------------------------------------------------------------------------------
```

Reference:
1.https://www.tensorflow.org/versions/r1.3/tutorials/wide_and_deep
2.https://github.com/tensorflow/models/tree/master/official/wide_deep
3.https://github.com/amygdala/tensorflow-workshop/blob/c62cfa3cd766cf0adf6d8fae7a289ae9e4ab161b/workshop_sections/wide_n_deep/wide_n_deep_flow2.ipynb
4.https://www.kaggle.com/jgmartin/deep-wide-model-prediction

Instruction

1. Recommandation environment: Anaconda virtual environment (python 3.6).
2. Install tensorflow 1.8.0 (newest version is 1.9.0).
3. Install all the other package. Install feature_selector for https://github.com/WillKoehrsen/feature-selector (https://github.com/WillKoehrsen/feature-selector). (put it in current directory).
4. Dataset from Kaggle competition https://www.kaggle.com/c/home-credit-default-risk/data (https://www.kaggle.com/c/home-credit-default-risk/data).

In [1]:

```
import os
import tempfile
import numpy as np
import tensorflow as tf
import pandas as pd
import gc
import time
from contextlib import contextmanager
from sklearn import preprocessing
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold, StratifiedKFold
#download feature_selector package from https://github.com/WillKoehrsen/feature-sele
from feature_selector import FeatureSelector
from sklearn.preprocessing import StandardScaler
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarn
ing: Conversion of the second argument of issubdtype from `float` to `
np.floating` is deprecated. In future, it will be treated as `np.float
64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

In [2]:

```
SRC_PATH = "./dataset/home_credit/sources"
OUT_PATH = "./dataset/home_credit/outputs"
print(os.listdir(SRC_PATH))
```

```
['application_test.csv', '.DS_Store', 'application_train.csv', 'sample
_submission.csv']
```

Functions

In [3]:

```
# Fill nan values
def fillna_df(df, verbose=False):
    cat_cols_object = df.dtypes[df.dtypes == 'object'].index
    cat_cols_int = df.dtypes[df.dtypes == 'int64'].index
    numeric_cols = df.dtypes[df.dtypes == 'float64'].index
    if verbose:
        display(get_misstable(df[cat_cols_object]))
        display(get_misstable(df[cat_cols_int]))
        display(get_misstable(df[numeric_cols]))
    df[cat_cols_object] = df[cat_cols_object].fillna('etc')
    df[cat_cols_int] = df[cat_cols_int].fillna(0)
    df[numeric_cols] = df[numeric_cols].fillna(0)
    return df, cat_cols_object, cat_cols_int, numeric_cols

def cross_validation(df,train_labels):
    from sklearn.model_selection import train_test_split
    train_X, val_X, train_y, val_y = train_test_split(df, train_labels, test_size=0.
    return train_X, val_X, train_y, val_y

def separate_columns(df):
    cate_columns = []
    num_columns = []
    #separate columns
    for column in df.columns:
        if column in list(df.select_dtypes(include=['object']).columns):
            cate_columns.append(column)
        if column in list(df.select_dtypes(exclude=['object']).columns):
            num_columns.append(column)
    return cate_columns,num_columns

def conv_feature_columns(df):
    cate_columns, num_columns=separate_columns(df)
    tf_num_feature_column=[]
    tf_cate_feature_column=[]
    for column in num_columns:
        column_name =str(column)
        column_name = tf.feature_column.numeric_column(column)
        tf_num_feature_column.append(column_name)
    for column in cate_columns:
        column_name =str(column)
        vocabulary_list_c=df[column].unique().tolist()
        column_name = tf.feature_column.categorical_column_with_vocabulary_list(colu
        tf_cate_feature_column.append(column_name)
    #hashing from categories to numerical use API
    #transformation using bucketized for numerical to categories use API
    return tf_num_feature_column,tf_cate_feature_column

def indicator_deep_column(tf_cate_feature_column):
    tf_cate_feature_column_indicator=[]
    for column in tf_cate_feature_column:
        column_indicator=tf.feature_column.indicator_column(column)
        tf_cate_feature_column_indicator.append(column_indicator)
    return tf_cate_feature_column_indicator

def cross_feature_selection(df):
    crossed_col=[]
    #1
    DAYS_BIRTH_c=tf.feature_column.bucketized_column(tf.feature_column.numeric_colu
                                                     boundaries = [-25000,-20000,-15000,-1
```

```
    CNT_CHILDREN_c=tf.feature_column.bucketized_column(tf.feature_column.numeric_co
                                                             boundaries = [2,4,6,8,10])
    crossed_col_1 = tf.feature_column.crossed_column( [DAYS_BIRTH_c, CNT_CHILDREN_c
    crossed_col.append(crossed_col_1)
    #2
    DEF_30_CNT_SOCIAL_CIRCLE_c=tf.feature_column.bucketized_column(tf.feature_colum
                                                                         boundaries = [2,4,6,8]
    OBS_30_CNT_SOCIAL_CIRCLE_c=tf.feature_column.bucketized_column(tf.feature_colum
                                                             boundaries = [5, 10,15,20,25,30])
    crossed_col_2 = tf.feature_column.crossed_column([DEF_30_CNT_SOCIAL_CIRCLE_c,OBS
    crossed_col.append(crossed_col_2)
#     #5 too much missing value
#     YEARS_BUILD_AVG_c=tf.feature_column.bucketized_column(tf.feature_column.numer
#                                                            boundaries = [0,0.2,0.4,0.6,0.8])
#     APARTMENTS_AVG_c=tf.feature_column.bucketized_column(tf.feature_column.numeri
#                                                              boundaries = [0,0.2,0.4,0.6,0.8]
#     crossed_col_5 = tf.feature_column.crossed_column( [YEARS_BUILD_AVG_c, APARTME
#     crossed_col.append(crossed_col_5)
#     #3
#     DEF_30_CNT_SOCIAL_CIRCLE_c2=tf.feature_column.bucketized_column(tf.feature_co
#                                                                         boundaries = [2,4,6,8
#     OBS_60_CNT_SOCIAL_CIRCLE_c=tf.feature_column.bucketized_column(tf.feature_col
#                                                              boundaries = [5, 10,15,20,25,30]
#     crossed_col_3 = tf.feature_column.crossed_column([DEF_30_CNT_SOCIAL_CIRCLE_c2
#     crossed_col.append(crossed_col_3)
#     #4
#     DAYS_REGISTRATION_c=tf.feature_column.bucketized_column(tf.feature_column.num
#                                                            boundaries = [-25000,-20000,-15000
#     DAYS_BIRTH_c2=tf.feature_column.bucketized_column(tf.feature_column.numeric_c
#                                                            boundaries = [-25000,-20000,-15000
#     crossed_col_4 = tf.feature_column.crossed_column( [DAYS_BIRTH_c2, DAYS_REGIST
#     crossed_col.append(crossed_col_4)
#     crossed_col = crossed_col_1 + crossed_col_2 + crossed_col_3 + crossed_col_4
    return crossed_col

def wide_deep_columns(df):
    tf_num_feature_column,tf_cate_feature_column=conv_feature_columns(df)
    deep_column_indicator_part = indicator_deep_column(tf_cate_feature_column)
    #categories in base_column
    base_column = tf_cate_feature_column
    #categories types with 0.3-0.7 cor
    crossed_column = []
    wide_column = []
    deep_column = []
    crossed_column=cross_feature_selection(df)
    wide_column = base_column + crossed_column
    deep_column = tf_num_feature_column + deep_column_indicator_part
    return wide_column,deep_column


def grid_selection(train,train_labels):
    fs = FeatureSelector(data = train, labels = train_labels)
    fs.identify_all(selection_params = {'missing_threshold': 0.6, 'correlation_thres
                                        'task': 'classification', 'eval_metric': 'auc',
                                         'cumulative_importance': 0.99})
    train_removed_all_once = fs.remove(methods = 'all', keep_one_hot = False)
    fs.feature_importances.head()
    fs.record_collinear.head()
    return train_removed_all_once
```

Load data and preprocessing and get feature columns

In [4]:

```
#load dataset
train = pd.read_csv(SRC_PATH + '/application_train.csv')
test= pd.read_csv(SRC_PATH + '/application_test.csv')



df=train
no_frauds = len(df[df['TARGET'] == 1])
non_fraud_indices = df[df.TARGET == 0].index
random_indices = np.random.choice(non_fraud_indices,no_frauds, replace=False)
fraud_indices = df[df.TARGET == 1].index
under_sample_indices = np.concatenate([fraud_indices,random_indices])
under_sample = df.loc[under_sample_indices]

train=under_sample
train_labels = train['TARGET']
#y_df = pd.Series(y, index=X.index)

#drop label and user Id columns
train = train.drop(columns = ['TARGET','SK_ID_CURR'])

#preprocessing
train_removed_all_once = grid_selection(train,train_labels)
df, cat_cols_object, cat_cols_int, numeric_cols=fillna_df(train_removed_all_once, ve

# test.columns = X.columns

#scalings
df_scale=df
scale_column=df_scale.select_dtypes(exclude=['object']).columns
scaler = StandardScaler().fit(df_scale[scale_column])
df_scale.loc[:,scale_column] = scaler.transform(df_scale[scale_column])

#get columns
wide_columns,deep_columns = wide_deep_columns(df_scale)

#train and validation separate
train_X, val_X, train_y, val_y = cross_validation(df_scale,train_labels)
```

```
23 features with greater than 0.60 missing values.

2 features with a single unique value.

32 features with a correlation magnitude greater than 0.95.

Training Gradient Boosting Model

Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[194]   valid_0's auc: 0.752953
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[158]   valid_0's auc: 0.755829
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[153]   valid_0's auc: 0.756684
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[201]   valid_0's auc: 0.762388
```

```
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[370]	valid_0's auc: 0.758138
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[146]	valid_0's auc: 0.762365
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[159]	valid_0's auc: 0.757728
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[117]	valid_0's auc: 0.751372
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[146]	valid_0's auc: 0.754786
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[195]	valid_0's auc: 0.756342

39 features with zero importance after one-hot encoding.

139 features required for cumulative importance of 0.99 after one hot
encoding.
103 features do not contribute to cumulative importance of 0.99.

143 total features out of 258 identified for removal after one-hot enc
oding.

['missing', 'single_unique', 'collinear', 'zero_importance', 'low_impo
rtance'] methods have been run

Removed 200 features including one-hot features.
```

In [5]:

```
sum(under_sample["TARGET"].values==1)
```

Out[5]:

```
24825
```

Config

In [12]:

```
model_dir ='./widendeep0'

run_config = tf.estimator.RunConfig(model_dir=model_dir,
                                    save_checkpoints_secs=300,
                                    keep_checkpoint_max=3)
```

In [13]:

```
estimator = tf.estimator.DNNLinearCombinedClassifier(linear_feature_columns=wide_col
                                                    dnn_hidden_units=[500,150,50], d
                                                    dnn_dropout=0.5,config=run_confi
# estimator = tf.estimator.DNNLinearCombinedClassifier(linear_feature_columns=wide_c
#                                                       dnn_hidden_units=[100,75,50],
#                                                       dnn_dropout=0.5,config=run_con

# estimator = tf.estimator.DNNLinearCombinedClassifier(linear_feature_columns=wide_c
#                                                       dnn_hidden_units=[500,250,150,
#                                                       linear_optimizer = tf.train.Ft
#                                                       dnn_dropout=0.5,config=run_con
# estimator = tf.estimator.DNNLinearCombinedClassifier(
#         model_dir=model_dir,
#         linear_feature_columns=wide_columns,
#         dnn_feature_columns=deep_columns,
#         dnn_hidden_units=[100, 75, 50,25],
#         config=run_config,
#         linear_optimizer = tf.train.FtrlOptimizer(learning_rate=0.0001,l1_regulari
#         dnn_optimizer=tf.train.ProximalAdagradOptimizer(0.000 1,initial_accumulato
```

```
INFO:tensorflow:Using config: {'_model_dir': './widendeep0', '_tf_rand
om_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps':
None, '_save_checkpoints_secs': 300, '_session_config': None, '_keep_c
heckpoint_max': 3, '_keep_checkpoint_every_n_hours': 10000, '_log_step
_count_steps': 100, '_train_distribute': None, '_service': None, '_clu
ster_spec': <tensorflow.python.training.server_lib.ClusterSpec object
at 0x1a213e37f0>, '_task_type': 'worker', '_task_id': 0, '_global_id_i
n_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chief': T
rue, '_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

input fn

In [14]:

```
train_input_fn = tf.estimator.inputs.pandas_input_fn(train_X, train_y, batch_size =
eval_input_fn = tf.estimator.inputs.pandas_input_fn(val_X, val_y, batch_size = 5000,
pred_input_fn = tf.estimator.inputs.pandas_input_fn(val_X, val_y, batch_size = len(v
```

Train and evaluation

In [15]:

```
train_spec = tf.estimator.TrainSpec(input_fn=train_input_fn, max_steps=10000, hooks=
eval_spec = tf.estimator.EvalSpec(input_fn=eval_input_fn, steps=10, start_delay_secs
                                  exporters=None, hooks=None)
```

In [16]:

```
%%time
tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

```
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-A
UCs; please switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-07-26-11:58:32
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ./widendeep0/model.ckpt-1000
0
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [1/10]
INFO:tensorflow:Evaluation [2/10]
INFO:tensorflow:Evaluation [3/10]
INFO:tensorflow:Finished evaluation at 2018-07-26-11:58:34
INFO:tensorflow:Saving dict for global step 10000: accuracy = 0.66422
3, accuracy_baseline = 0.50237656, auc = 0.7217915, auc_precision_reca
ll = 0.70077676, average_loss = 0.6185109, global_step = 10000, label/
mean = 0.49762347, loss = 2559.1921, precision = 0.66507804, predictio
n/mean = 0.4950393, recall = 0.6551724
CPU times: user 7min 5s, sys: 13 s, total: 7min 18s
Wall time: 5min 27s
```

In [17]:

```
valid_results = estimator.evaluate(input_fn=pred_input_fn, steps=1)
print()
print("-----------------------------------------------------------------------------------
print(format(valid_results))
print("-----------------------------------------------------------------------------------
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-A
UCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-A
UCs; please switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-07-26-11:58:37
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ./widendeep0/model.ckpt-1000
0
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [1/1]
INFO:tensorflow:Finished evaluation at 2018-07-26-11:58:40
INFO:tensorflow:Saving dict for global step 10000: accuracy = 0.66422
3, accuracy_baseline = 0.50237656, auc = 0.7217915, auc_precision_reca
ll = 0.70077676, average_loss = 0.61851084, global_step = 10000, labe
l/mean = 0.49762347, loss = 7677.575, precision = 0.66507804, predicti
on/mean = 0.4950392, recall = 0.6551724

----------------------------------------------------------------------
---------------------------------------
{'accuracy': 0.664223, 'accuracy_baseline': 0.50237656, 'auc': 0.72179
15, 'auc_precision_recall': 0.70077676, 'average_loss': 0.61851084, 'l
abel/mean': 0.49762347, 'loss': 7677.575, 'precision': 0.66507804, 'pr
ediction/mean': 0.4950392, 'recall': 0.6551724, 'global_step': 10000}
----------------------------------------------------------------------
---------------------------------------
```

Prediction

In [18]:

```
column_name=list(df)
```

In [19]:

```
test_column_reduce=test[column_name]
```

In [20]:

```
test_final,cat_cols_object_test, cat_cols_int_test, numeric_cols_test=fillna_df(test
```

```
/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py:3137: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  self[k1] = value[k2]
```

In [21]:

```
#test_final
```

In [22]:

```
pred_input_fn_test = tf.estimator.inputs.pandas_input_fn(test_final,batch_size = 128
```

In [23]:

```
predictions = list(estimator.predict(input_fn=pred_input_fn_test))
predicted_classes = [p["classes"] for p in predictions]
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ./widendeep0/model.ckpt-1000
0
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

In [24]:

```
# predictions = estimator.predict(input_fn=pred_input_fn_test)
# df_pred = pd.DataFrame(predictions)
# sample = pd.read_csv(SRC_PATH + '/sample_submission.csv')
# sample['TARGET'] = df_pred.values
# sample.to_csv('results.csv')
```

In [25]:

```
print(
    "New Samples, Class Predictions:    {}\n"
    .format(predicted_classes))
```

```
pe=object), array([b'0'], dtype=object), array([b'0'], dtype=object),
 array([b'0'], dtype=object), array([b'0'], dtype=object), array
([b'0'], dtype=object), array([b'0'], dtype=object), array([b'0'], dty

pe=object), array([b'0'], dtype=object), array([b'0'], dtype=object),
 array([b'0'], dtype=object), array([b'0'], dtype=object), array
([b'0'], dtype=object), array([b'0'], dtype=object), array([b'0'], dty
pe=object), array([b'0'], dtype=object), array([b'0'], dtype=object),
 array([b'0'], dtype=object), array([b'0'], dtype=object), array
([b'0'], dtype=object), array([b'0'], dtype=object), array([b'0'], dty
pe=object), array([b'0'], dtype=object), array([b'0'], dtype=object),
 array([b'0'], dtype=object), array([b'0'], dtype=object), array
([b'0'], dtype=object), array([b'0'], dtype=object), array([b'0'], dty
pe=object), array([b'0'], dtype=object), array([b'0'], dtype=object),
 array([b'0'], dtype=object), array([b'0'], dtype=object), array
([b'0'], dtype=object), array([b'0'], dtype=object), array([b'0'], dty
pe=object), array([b'0'], dtype=object), array([b'0'], dtype=object),
 array([b'0'], dtype=object), array([b'0'], dtype=object), array
([b'0'], dtype=object), array([b'0'], dtype=object), array([b'0'], dty
pe=object), array([b'0'], dtype=object), array([b'0'], dtype=object),
```

In [26]:

```
def get_score_from_estimator(estimator, input_fn, size):
    import itertools
    score = estimator.predict(input_fn=input_fn)
    predictions = list(itertools.islice(score, size))
    print("the key of predictions = ", list(predictions[0].keys()))
    score = np.array([dic['logistic'] for dic in predictions])
    return score
```

In [27]:

```
%%time

score = get_score_from_estimator(estimator, pred_input_fn, len(val_X))
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ./widendeep0/model.ckpt-1000
0
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
the key of predictions =  ['logits', 'logistic', 'probabilities', 'cla
ss_ids', 'classes']
CPU times: user 4.07 s, sys: 81.8 ms, total: 4.15 s
Wall time: 3.94 s
```

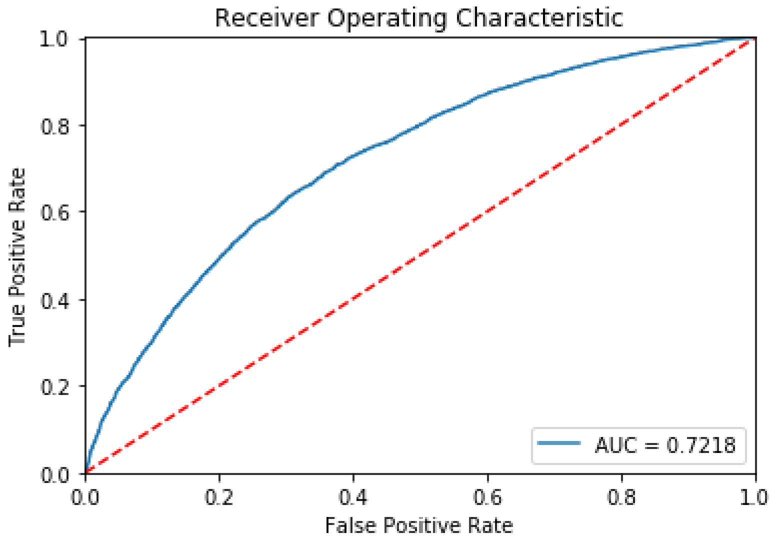
In [28]:

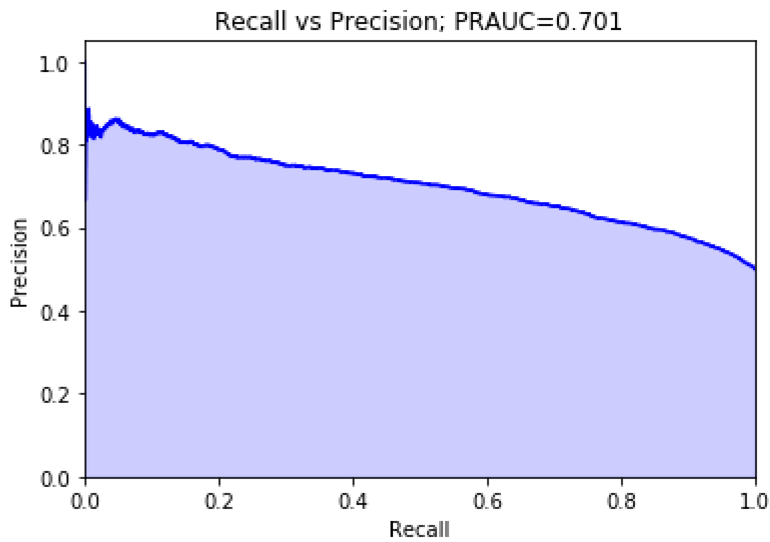
```
# Evaluation metrics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import auc
from sklearn.metrics import (confusion_matrix, classification_report,
                             roc_curve, average_precision_score, precision_recall_cu
                             precision_score, recall_score, f1_score, matthews_corr


def plot_roc_curve(y_true, y_score):
    '''
    Plot ROC Curve
    '''
    fpr, tpr, thresholds = roc_curve(y_true, y_score)
    roc_auc = auc(fpr, tpr)

    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, label='AUC = %0.4f' % roc_auc)
    plt.legend(loc='lower right')
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlim([-0.001, 1])
    plt.ylim([0, 1.001])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()


def plot_pr_curve(y_true, y_score):
    '''
    Plot Precision-recall Curve
    '''
    precision, recall, th = precision_recall_curve(y_true, y_score)
    avg_prec = average_precision_score(y_true, y_score)

    plt.step(recall, precision, color='b', alpha=0.2, where='post')
    plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')

    plt.plot(recall, precision, 'b', label='Precision-Recall curve')
    plt.title('Recall vs Precision; PRAUC={0:0.3f}'.format(avg_prec))
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.show()


def plot_conf_mtx(y_true, y_pred, class_labels):
    '''
    Plot Confusion matrix
    '''
    print(classification_report(y_true, y_pred, target_names=class_labels))
    conf_matrix = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(4, 4))
    sns.heatmap(conf_matrix, xticklabels=class_labels, yticklabels=class_labels, ann
    plt.title("Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()
```

```
def get_binary_classification_metrics(y_true, y_score, y_pred, ndigits=3):
    fpr, tpr, _ = roc_curve(y_true, y_score)
    auroc = round(auc(fpr, tpr), ndigits)
    auprc = round(average_precision_score(y_true, y_score), ndigits)
    mcc = round(matthews_corrcoef(y_true, y_pred), ndigits)
    prec = round(precision_score(y_true, y_pred), ndigits)
    rec = round(recall_score(y_true, y_pred), ndigits)
    f1 = round(f1_score(y_true, y_pred), ndigits)

    metrics = {'AUROC': [auroc], 'AUPRC': [auprc], 'MCC': [mcc],
               'Precision': [prec], 'Recall': [rec], 'F1': [f1]}
    metrics = pd.DataFrame(metrics, columns=['AUROC', 'AUPRC', 'MCC', 'Precision',

    return metrics

def eval_result(y_true, score, cutoff_pred=0.5):
    y_pred = score > cutoff_pred
    plot_roc_curve(y_true, score)
    plot_pr_curve(y_true, score)
    class_labels = ["Majority", "Minority"]
    plot_conf_mtx(y_true, y_pred, class_labels)
    metrics = get_binary_classification_metrics(y_true, score, y_pred)
    display(metrics)
```

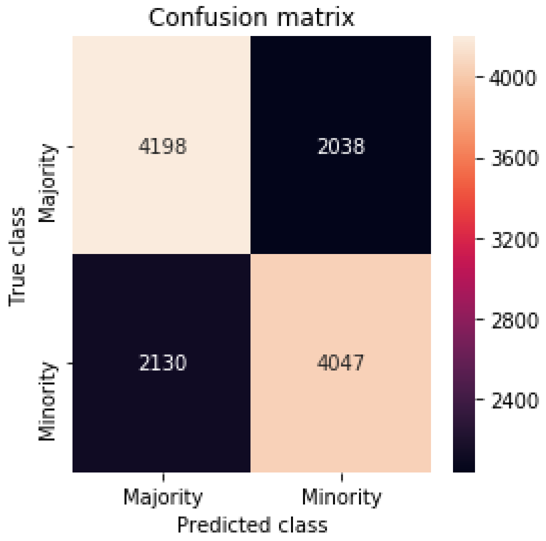
In [29]:

```
eval_result(val_y.values, score)
```






```
             precision    recall  f1-score   support

   Majority       0.66      0.67      0.67      6236
   Minority       0.67      0.66      0.66      6177

avg / total       0.66      0.66      0.66     12413
```



	AUROC	AUPRC	MCC	Precision	Recall	F1
0	0.722	0.701	0.328	0.665	0.655	0.66

Experiment on crossed feature columns selected

In [*]:

```
fs = FeatureSelector(data = train, labels = train_labels)
fs.identify_all(selection_params = {'missing_threshold': 0.6, 'correlation_threshold
                                    'task': 'classification', 'eval_metric': 'auc',
                                     'cumulative_importance': 0.99})
```

```
23 features with greater than 0.60 missing values.

2 features with a single unique value.

59 features with a correlation magnitude greater than 0.30.

Training Gradient Boosting Model

Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[148]   valid_0's auc: 0.749542
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[146]   valid_0's auc: 0.752516
Training until validation scores don't improve for 100 rounds.
```

Missing value < 60%

In [ ]:

```
fs.identify_missing(missing_threshold = 0.6)
```

In [ ]:

```
miss_df=fs.missing_stats
```

In [ ]:

```
miss_df_above=miss_df[miss_df['missing_fraction']<0.6]
```

In [ ]:

```
#miss_df_above
```

Feature importance Top 50

In [ ]:

```
#fs.feature_importances.head(50)
```

In [ ]:

```
# fs.record_collinear.head()
```

In [ ]:

```
fea_importance=fs.feature_importances.head(50)
```

In [ ]:

```
list_fea_importance=fea_importance['feature']
```

In [ ]:

```
#list_fea_importance
```

In [ ]:

```
#fs.record_collinear
```

Select corr 0.3-0.4

In [ ]:

```
df_pre_cross = fs.record_collinear
```

In [ ]:

```
df2=df_pre_cross[df_pre_cross['corr_value']>=0.3]
```

In [ ]:

```
df3=df2[df2['corr_value']<=0.4]
```

In [ ]:

```
#df3
```

In [ ]:

```
df4=df3[df3['drop_feature'].isin(list_fea_importance)]
```

In [ ]:

```
df5=df4[df4['drop_feature'].isin(list_fea_importance)]
```

In [ ]:

```
crossed_fea_A=df4['drop_feature']
crossed_fea_B=df4['corr_feature']
```

Creat crossed column

In [ ]:

```
YEARS_BUILD_AVG_c=tf.feature_column.bucketized_column(tf.feature_column.numeric_colu
                                                boundaries = [0,0.2,0.4,0.6,0.8])
APARTMENTS_AVG_c=tf.feature_column.bucketized_column(tf.feature_column.numeric_colur
                                                boundaries = [0,0.2,0.4,0.6,0.8])
crossed_col_5 = tf.feature_column.crossed_column( [YEARS_BUILD_AVG_c, APARTMENTS_AV(
```

In [ ]:

```
DAYS_BIRTH=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('DAY
                                                boundaries = [-25000,-20000,-15000,-1
CNT_CHILDREN=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('(
                                                boundaries = [2,4,6,8,10])
crossed_col_1 = tf.feature_column.crossed_column( [DAYS_BIRTH, CNT_CHILDREN], 5000)
```

In [ ]:

```
EF_30_CNT_SOCIAL_CIRCLE=tf.feature_column.bucketized_column(tf.feature_column.numeri(
                                                boundaries = [2,4,6,8])
BS_30_CNT_SOCIAL_CIRCLE=tf.feature_column.bucketized_column(tf.feature_column.numeri(
                                                boundaries = [5, 10,15,20,25,30])
rossed_col_2 = tf.feature_column.crossed_column([DEF_30_CNT_SOCIAL_CIRCLE,OBS_30_CNT_
```

In [ ]:

```
DEF_30_CNT_SOCIAL_CIRCLE=tf.feature_column.bucketized_column(tf.feature_column.numer
                                                boundaries = [2,4,6,8])
OBS_60_CNT_SOCIAL_CIRCLE=tf.feature_column.bucketized_column(tf.feature_column.numer
                                                boundaries = [5, 10,15,20,25,30])
crossed_col_3 = tf.feature_column.crossed_column([DEF_30_CNT_SOCIAL_CIRCLE,OBS_30_CI
```

In [ ]:

```
DAYS_REGISTRATION=tf.feature_column.bucketized_column(tf.feature_column.numeric_colu
                                                boundaries = [-25000,-20000,-15000,-1
DAYS_BIRTH=tf.feature_column.bucketized_column(tf.feature_column.numeric_column('DAY
                                                boundaries = [-25000,-20000,-15000,-1
crossed_col_4 = tf.feature_column.crossed_column( [DAYS_BIRTH, DAYS_REGISTRATION], 5
```