Wide and deep neural network

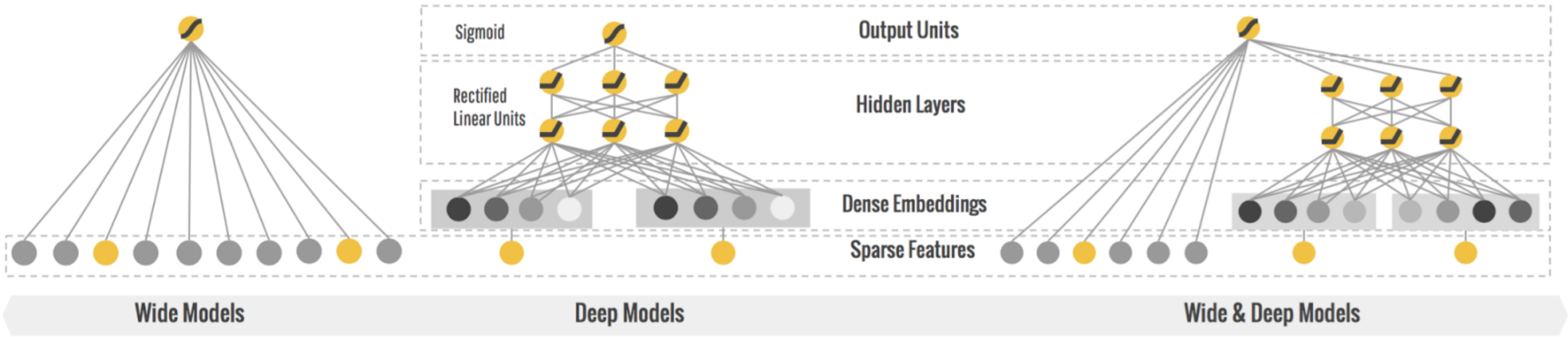
Hyundai Card Internship Program
2018 Summer
Siqi Shao

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

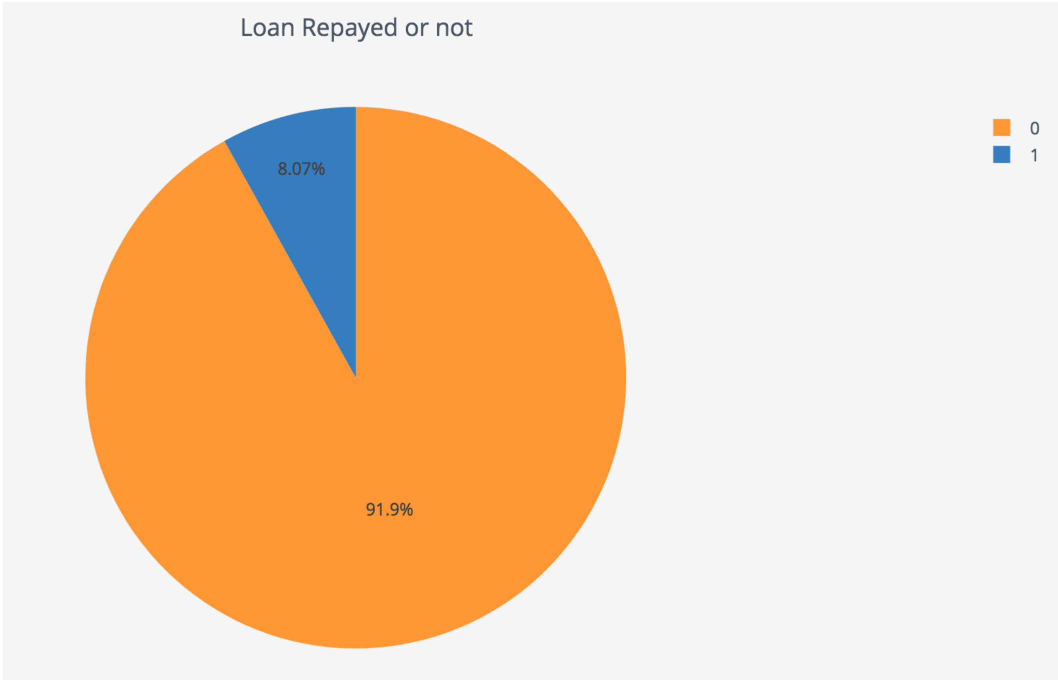
Home Credit Group is a credit card company and predicting whether or not a client will repay a loan or have difficulty is a critical business need for them.

• 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
correlation calculation

Exploratory data analysis(EDA)

↓

Filling missing value – Preprocessing – Scaling

↓

Remove some features – Feature Selection

Build feature columns

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)
```

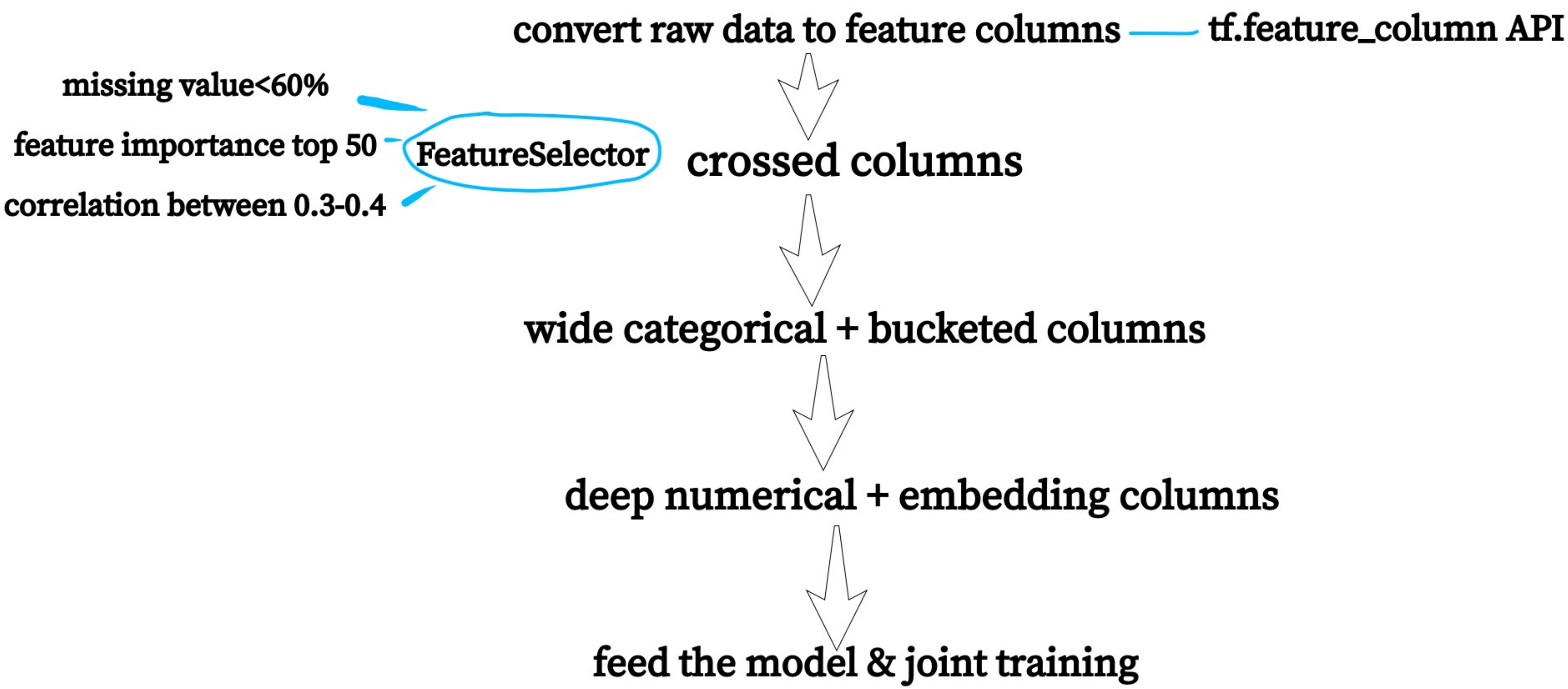
```
['missing', 'single_unique', 'collinear', 'zero_importance', 'low_importance'] methods have been r
un
```

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

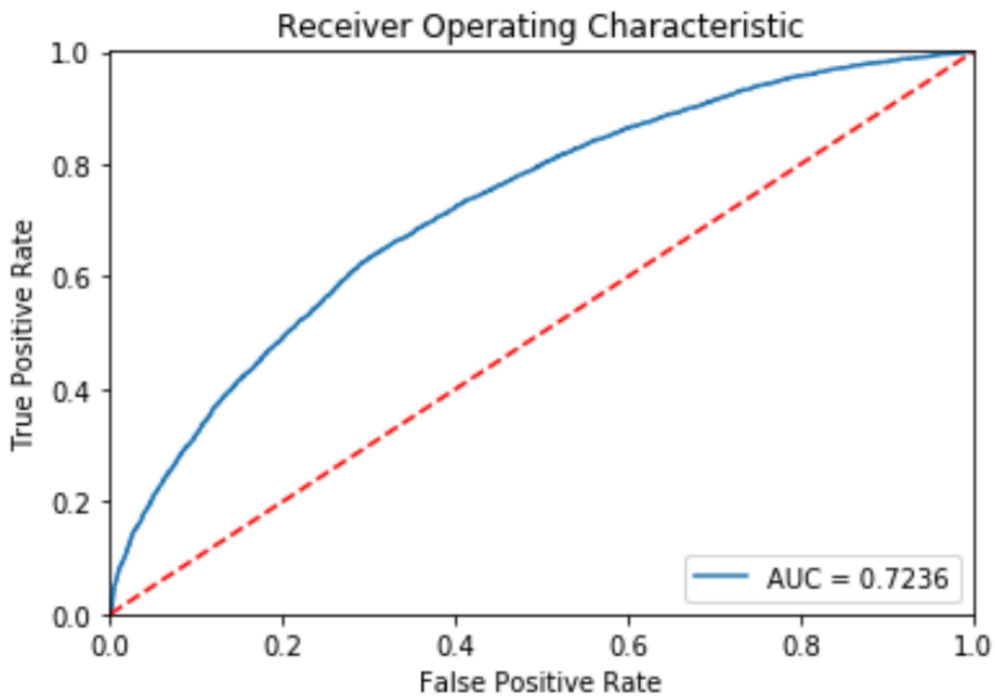
- wide part = sparse columns + crossed columns
- deep part = continuous columns + embedded categorical column
- Using hashing and bucketizing to transform between categorical and numerical data.
- Joint train Wide & Deep model (`DNNLinearCombinedClassifier`).

```
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')

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)
```

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

0 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:
[278]   valid_0's auc: 0.76186
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[496]   valid_0's auc: 0.756064
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[387]   valid_0's auc: 0.759601
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[385]   valid_0's auc: 0.761217
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[296]   valid_0's auc: 0.754435
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[265]   valid_0's auc: 0.764486
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[283]   valid_0's auc: 0.757353
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
```

```
[297]   valid_0's auc: 0.757855
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[259]   valid_0's auc: 0.765589
Training until validation scores don't improve for 100 rounds.
Early stopping, best iteration is:
[385]   valid_0's auc: 0.758508

23 features with zero importance after one-hot encoding.

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

125 total features out of 260 identified for removal after one-hot enc
oding.

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

Removed 194 features including one-hot features.
```

Config

In [29]:

```
model_dir ='./widendeep'

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

In [30]:

```
estimator = tf.estimator.DNNLinearCombinedClassifier(linear_feature_columns=wide_co
                                                     dnn_hidden_units=[500,150,50], 
                                                     dnn_dropout=0.5,config=run_conf
# estimator = tf.estimator.DNNLinearCombinedClassifier(linear_feature_columns=wide_
#                                                      dnn_hidden_units=[100,75,50],
#                                                      dnn_dropout=0.5,config=run_co

# estimator = tf.estimator.DNNLinearCombinedClassifier(linear_feature_columns=wide_
#                                                      dnn_hidden_units=[500,250,150
#                                                      linear_optimizer = tf.train.F
#                                                      dnn_dropout=0.5,config=run_co
# 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_regular
#         dnn_optimizer=tf.train.ProximalAdagradOptimizer(0.000 1,initial_accumulat
```

```
INFO:tensorflow:Using config: {'_model_dir': './widendeep13', '_tf_ran
dom_seed': None, '_save_summary_steps': 100, '_save_checkpoints_step
s': None, '_save_checkpoints_secs': 300, '_session_config': None, '_ke
ep_checkpoint_max': 3, '_keep_checkpoint_every_n_hours': 10000, '_log_
step_count_steps': 100, '_train_distribute': None, '_service': None,
'_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec ob
ject at 0x1a15fa2dd8>, '_task_type': 'worker', '_task_id': 0, '_global
_id_in_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chie
f': True, '_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

input fn

In [31]:

```
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(
```

Train and evaluation

In [32]:

```
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_sec
                                  exporters=None, hooks=None)
```

In [33]:

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

```
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [1/10]
INFO:tensorflow:Evaluation [2/10]
INFO:tensorflow:Evaluation [3/10]
INFO:tensorflow:Evaluation [4/10]
INFO:tensorflow:Evaluation [5/10]
INFO:tensorflow:Evaluation [6/10]
INFO:tensorflow:Evaluation [7/10]
INFO:tensorflow:Evaluation [8/10]
INFO:tensorflow:Evaluation [9/10]
INFO:tensorflow:Evaluation [10/10]
INFO:tensorflow:Finished evaluation at 2018-07-25-01:44:01
INFO:tensorflow:Saving dict for global step 100000: accuracy = 0.9195
4, accuracy_baseline = 0.91954, auc = 0.7260101, auc_precision_recall
= 0.19395277, average_loss = 0.25687274, global_step = 100000, label/m
ean = 0.08046, loss = 1284.3638, precision = 0.0, prediction/mean = 0.
090976395, recall = 0.0
CPU times: user 52min 16s, sys: 3min 12s, total: 55min 29s
Wall time: 29min 33s
```