# 111

February 13, 2021

[1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import warnings
import os
import DeepFM as dfm
import Preprocess as prep
from sklearn.metrics import␣
↪roc_curve,confusion_matrix,recall_score,roc_auc_score
import matplotlib.pyplot as plt
import datetime
warnings.filterwarnings('ignore')
path_model='D:\\Github\\projects-1\\DeepFM\\'
```

[17]:
```
train=pd.read_csv('d:\\Github\\projects-1\\Dataset\\train.csv')
item_pool=pd.read_csv('d:\\Github\\projects-1\\Dataset\\item_pool.csv')
#item_pool=list(item_pool['oper_obj'])
test=pd.read_csv('d:\\Github\\projects-1\\Dataset\\test.csv')
```

[3]:
```
train_set,test_set=prep.
↪process_data(train,item_pool,test,batch_size=16384,sampling_ratio=4)
```

[8]:
```
#log_dir=path_model+datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
#tensorboard_callback=tf.keras.callbacks.
↪TensorBoard(log_dir=log_dir,histogram_freq=1)
model=dfm.DeepFatorizationMachine(256,5000)
if os.path.exists(path_model+'DeepFM.h5'):
    print('loading model.\n')
    model.predict(test_set.take(1))
    model.load_weights(path_model+'DeepFM.h5')
    model.compile(loss=tf.keras.losses.BinaryCrossentropy(),optimizer=tf.keras.
↪optimizers.Adam(0.01),metrics=
     [tf.keras.metrics.BinaryAccuracy(),dfm.roc_auc,tf.keras.metrics.Recall()])
    model.fit(test_set,epochs=5)
    model.evaluate(test_set)
    model.save_weights(path_model+'DeepFM.h5')
else:
    model.compile(loss=tf.keras.losses.BinaryCrossentropy(),optimizer=tf.keras.
↪optimizers.Adam(0.01),metrics=
    [tf.keras.metrics.BinaryAccuracy(),dfm.roc_auc,tf.keras.metrics.Recall()])
    model.fit(train_set,epochs=20,validation_data=test_set)
    model.evaluate(test_set)
    model.summary()
    model.save_weights(path_model+'DeepFM.h5')
```

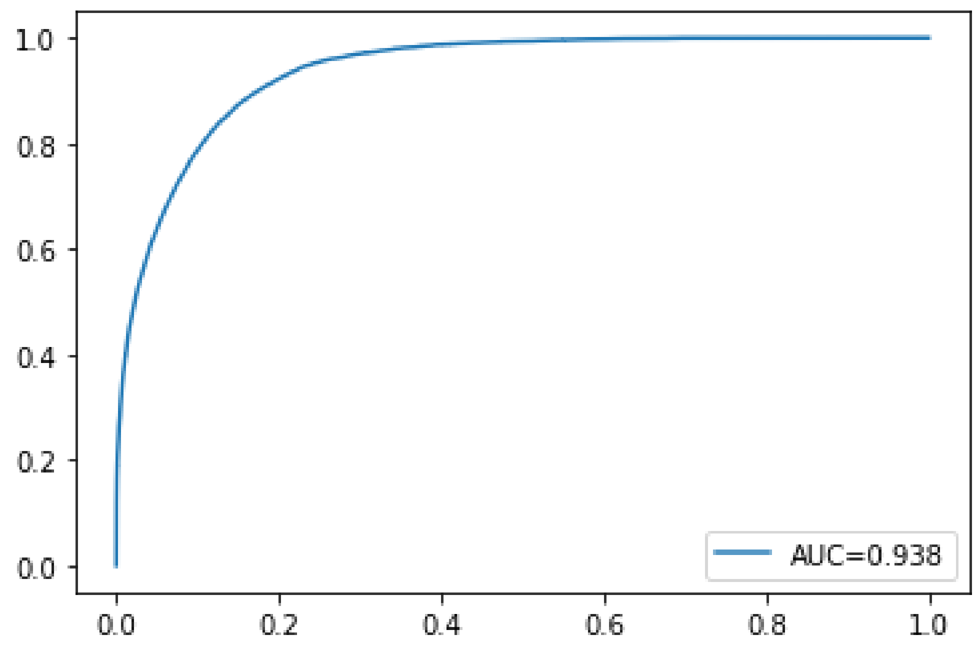
```
loading model.

Epoch 1/5
3/3 [==============================] - 1s 107ms/step - loss: 0.4155 -
binary_accuracy: 0.8627 - roc_auc: 0.9132 - recall_4: 0.4543
Epoch 2/5
3/3 [==============================] - 0s 107ms/step - loss: 0.3836 -
binary_accuracy: 0.8750 - roc_auc: 0.9242 - recall_4: 0.5444
Epoch 3/5
3/3 [==============================] - 0s 108ms/step - loss: 0.3160 -
binary_accuracy: 0.8804 - roc_auc: 0.9258 - recall_4: 0.5755
Epoch 4/5
3/3 [==============================] - 0s 107ms/step - loss: 0.3278 -
binary_accuracy: 0.8834 - roc_auc: 0.9303 - recall_4: 0.5766
Epoch 5/5
3/3 [==============================] - 0s 107ms/step - loss: 0.2973 -
binary_accuracy: 0.8844 - roc_auc: 0.9335 - recall_4: 0.5693
WARNING:tensorflow:5 out of the last 73 calls to <function
Model.make_test_function.<locals>.test_function at 0x0000024689246318> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
3/3 [==============================] - 0s 66ms/step - loss: 0.2825 -
binary_accuracy: 0.8890 - roc_auc: 0.9391 - recall_4: 0.6205
```

[9]:
```
y_true,y_score=dfm.get_prediction(model,test_set)
fpr,tpr,thresholds=roc_curve(y_true,y_score,drop_intermediate=False)
auc=roc_auc_score(y_true,y_score)
plt.plot(fpr,tpr)
plt.legend(['AUC='+str(np.round(auc,3))])
plt.show()
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x0000024688A77948>
```

```
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```



[10]:
```python
y_pred=np.zeros(len(y_true))
threshold=0.5
y_pred[y_score[:,0]>threshold]=1
```

[11]:
```python
recall_score(y_true,y_pred)
```

[11]:
```
0.6204556471158507
```

[26]:
```python
help(pd.DataFrame.sample)
```

```
Help on function sample in module pandas.core.generic:

sample(self, n=None, frac=None, replace=False, weights=None, random_state=None,
axis=None)
    Return a random sample of items from an axis of object.

    You can use `random_state` for reproducibility.
```

```
Parameters
----------
n : int, optional
    Number of items from axis to return. Cannot be used with `frac`.
    Default = 1 if `frac` = None.
frac : float, optional
    Fraction of axis items to return. Cannot be used with `n`.
replace : bool, default False
    Sample with or without replacement.
weights : str or ndarray-like, optional
    Default 'None' results in equal probability weighting.
    If passed a Series, will align with target object on index. Index
    values in weights not found in sampled object will be ignored and
    index values in sampled object not in weights will be assigned
    weights of zero.
    If called on a DataFrame, will accept the name of a column
    when axis = 0.
    Unless weights are a Series, weights must be same length as axis
    being sampled.
    If weights do not sum to 1, they will be normalized to sum to 1.
    Missing values in the weights column will be treated as zero.
    Infinite values not allowed.
random_state : int or numpy.random.RandomState, optional
    Seed for the random number generator (if int), or numpy RandomState
    object.
axis : int or string, optional
    Axis to sample. Accepts axis number or name. Default is stat axis
    for given data type (0 for Series and DataFrames).

Returns
-------
Series or DataFrame
    A new object of same type as caller containing `n` items randomly
    sampled from the caller object.

See Also
--------
numpy.random.choice: Generates a random sample from a given 1-D numpy
    array.

Examples
--------
>>> df = pd.DataFrame({'num_legs': [2, 4, 8, 0],
…                      'num_wings': [2, 0, 0, 0],
…                      'num_specimen_seen': [10, 2, 1, 8]},
…                     index=['falcon', 'dog', 'spider', 'fish'])
>>> df
```

```
        num_legs  num_wings  num_specimen_seen
falcon         2          2                 10
dog            4          0                  2
spider         8          0                  1
fish           0          0                  8

Extract 3 random elements from the ``Series`` ``df['num_legs']``:
Note that we use `random_state` to ensure the reproducibility of
the examples.

>>> df['num_legs'].sample(n=3, random_state=1)
fish      0
spider    8
falcon    2
Name: num_legs, dtype: int64

A random 50% sample of the ``DataFrame`` with replacement:

>>> df.sample(frac=0.5, replace=True, random_state=1)
      num_legs  num_wings  num_specimen_seen
dog          4          0                  2
fish         0          0                  8

Using a DataFrame column as weights. Rows with larger value in the
`num_specimen_seen` column are more likely to be sampled.

>>> df.sample(n=2, weights='num_specimen_seen', random_state=1)
        num_legs  num_wings  num_specimen_seen
falcon         2          2                 10
fish           0          0                  8
```

[3]:
```
train.head()
```

[3]:
```
                            user_id                           item_id
0  f62daadb2c10409ab319314e4fcd6d15  33a9bc0a35234b9d96206aca5ed78fdd
1  f62daadb2c10409ab319314e4fcd6d15  0ecbd81b65974328845f2687fa41e908
2  fffbee1a9969448fab86cfdcaad0e795  bba2ab7de2d34dfca5c9dd55c4999828
3  f62daadb2c10409ab319314e4fcd6d15  1cb48bf7673040d292ffa55cc1931fd9
4  f62daadb2c10409ab319314e4fcd6d15  53611dbb2d0b420482fa04e17261789a
```

[10]:
```
hash=tf.keras.layers.experimental.preprocessing.Hashing(num_bins=3)(train.iloc[:
 ↪5,:].values)
emb=tf.keras.layers.Embedding(input_dim=3,output_dim=64)(hash)
```

[13]:
```
tf.keras.layers.Flatten()(emb).shape
```

[13]:
```
TensorShape([5, 128])
```

[19]:
```
len(dict(train.groupby(['item_id'])['user_id'].count().
 ↪sort_values(ascending=False)))
```

[19]:
```
90
```

[18]:
```
item_pool.nunique()
```

[18]:
```
oper_obj    90
dtype: int64
```

[ ]:
```
tf.linalg.norm()
```