Wide & Deep Model

1、相关资料

- 英文paper：https://arxiv.org/pdf/1606.07792.pdf
- 中文资料：
  - 理论：https://zhuanlan.zhihu.com/p/34676942
  - 实现：https://blog.csdn.net/m0_37744293/article/details/69950262

2、数学原理

> **本质上，**Wide & Deep Model **是基于带交互的**LR + DNN **模型。**

- What is Wide & Deep

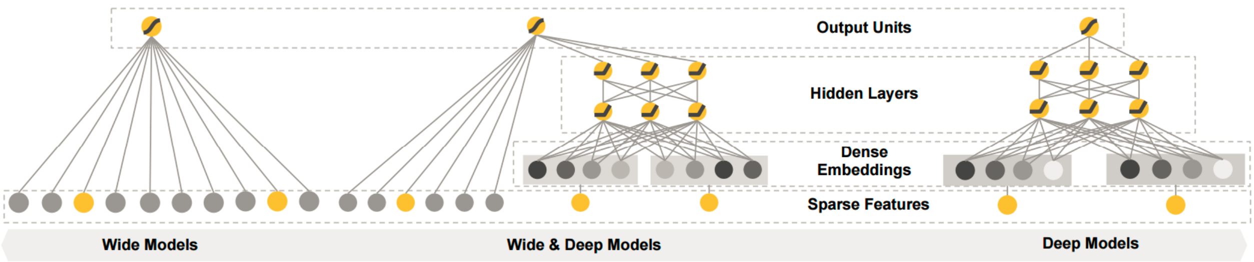

Figure 1: The spectrum of Wide & Deep models.

- 如何用Wide & Deep 玩转Recommender system

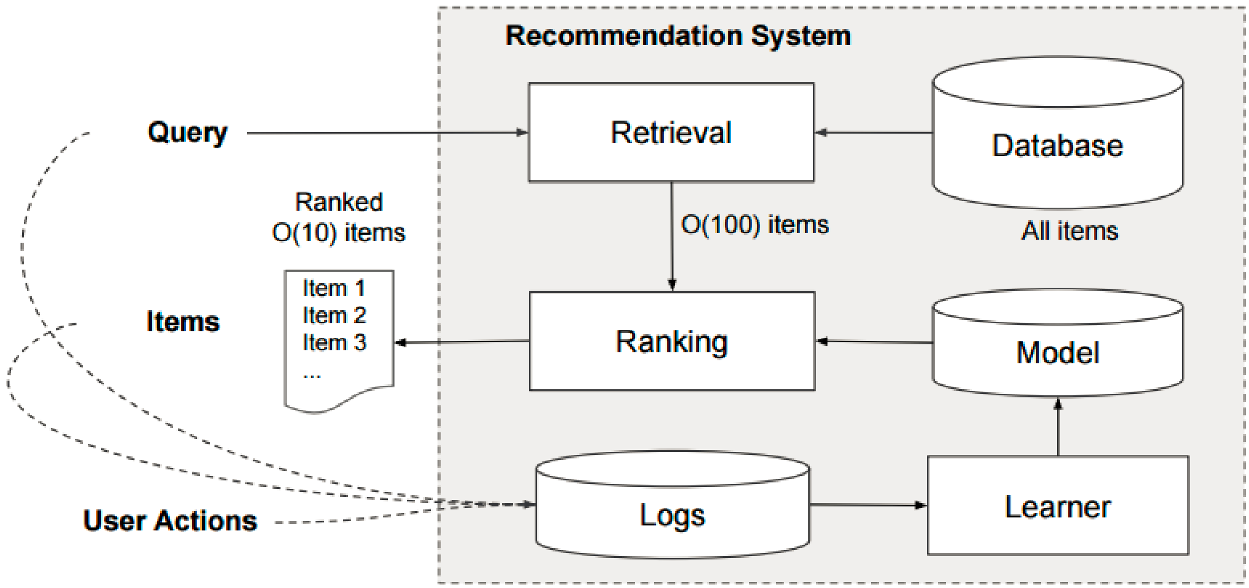

Figure 2: Overview of the recommender system.

- Recommender system pipeline overview

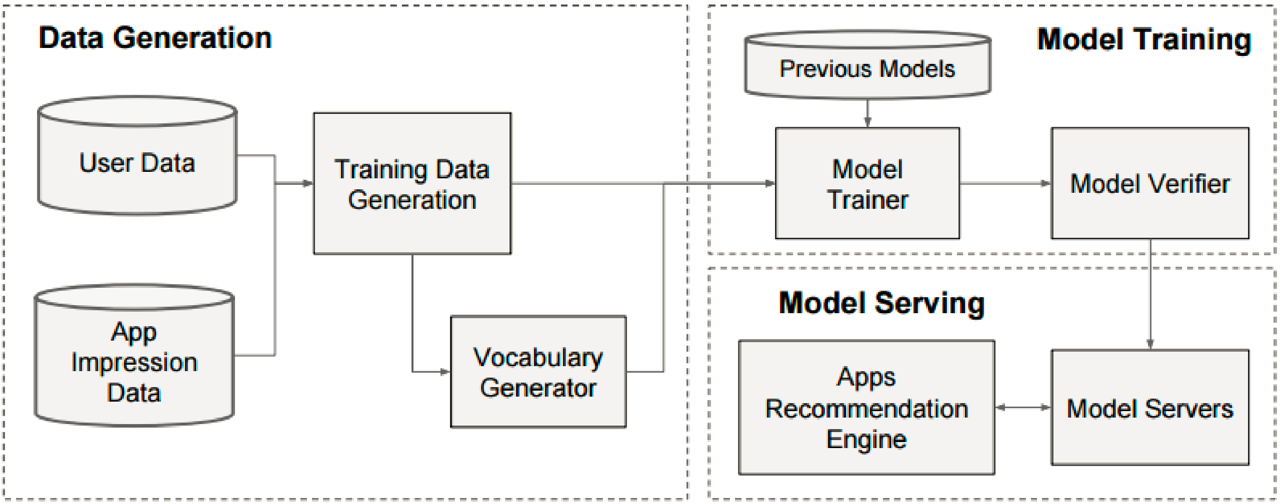

Figure 3: Apps recommendation pipeline overview.

- How Does Google Play Wide & Deep in Recommender system?

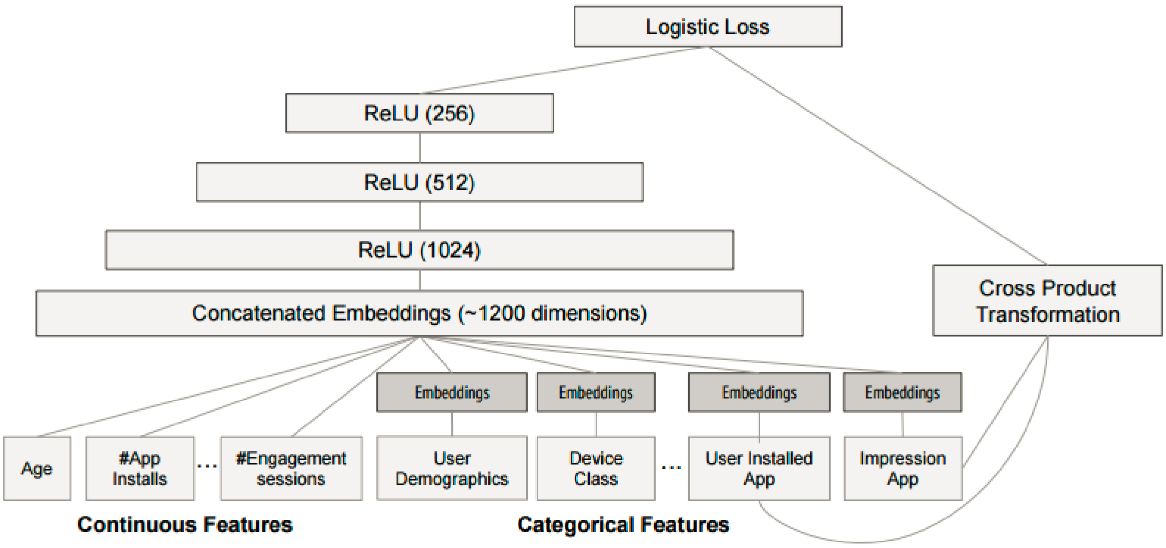

Figure 4: Wide & Deep model structure for apps recommendation.

- Wide
  - Wide组件解决“**记忆性**”的问题：比如通过历史数据知道”麻雀会飞”，”鸽子会飞”。
  - 由一个LR模型构成，特征集包括原始特征和交互特征。这里有一个坑：看paper会误以为模型默认是对所有one-hot后的特征进行交互。**但看**TensorFlow**的实现代码会发现：哪些**one-hot**后的特征需要进行交互，和谁进行交互，做几次幂的交互，均由**algo engineer**指定。**，具体公式如下：

    $\phi_k(x) = \prod_{i=1}^{d} x_i^{c_{ki}}, c_{ki} \in \{0, 1\}$
  - 如果历史数据里没有“钢铁侠会飞”，那么单靠Wide 组件就无法学到这部分知识。这也是LR本身的缺陷。
  - 具体做法：
    - 将分类特征做完one-hot后再进行cross product。
    - 个人理解，这里一方面是利用类似于FM模型原理来增强分类特征的特征交互（co-occurrence），另一方面是利用LR对高维稀疏特征的学习能力。
    - 作者把Wide Model所具备的能力称为“memorization”

- Deep
    - Deep组件解决“**泛化性**”的问题：从历史数据中从未见过的情形，比如”带翅膀的动物会飞”。
    - Deep 端对应的是 DNN 模型，每个特征对应一个低维的实数向量，我们称之为特征的embedding。DNN 模型通过反向传播（BP算法）调整隐藏层的权重，并且更新特征的 embedding。【其实整个 Wide & Deep Model 都是根据BP来更新参数的。】
    - Deep Model则是一个DNN，特征上除了原始特征还增加了分类特征的embedding，这个embedding在模型中属于独立的一层，embedding后的向量也是通过不断迭代学习出来的。
    - 将高维稀疏分类特征映射到低维embedding特征这种方式有助于模型进行“generalization”。
    - 每个 hideen layer都会做以下运算：
      $a^{l+1} = f(W^{(l)}a^{(l)} + b^{(l)})$
      l：层数
      f：激活函数（通常为$ReLUs$）
      $a^{(l)}$：第l层的$activations$，
      $b^{(l)}$：第l层的$bias$
      $W^{(l)}$：第l层的$weights$
    - [Why does DNN always use ReLUs?]()

- Wide & Deep
    - 一个是通过特征交互关系来训练浅层模型，另一个则是通过特征在映射空间中的信息训练深层模型。
    - 模型预测正例的公式为：
      $P(Y=1|x) = \sigma(w_{wide}^T[x, \phi(x)] + w_{deep}^T a^{(l_f)} + b)$
      其中Y是二分类的label，$\sigma(\cdot)$是sigmoid function，$\phi(x)$是对原始特征x做cross product transformations，b是bias项。w_{wide}是所有wide模型权重向量，w_{deep}是应用在最终激活函数$a^{(l_f)}$上的权重。

- Train Model
    - 这里有一个坑是：正确的最后一步是，wide组件和deep组件的输出的对数几率进行加权求和后，输入到共同的一个logistic loss function。paper原文内容：

      > The wide component and deep component are combined using a weighted sum of their output log odds as the prediction, which is then fed to one common logistic loss function for joint training.

    - 复习一下 logistic loss function:
      $cost(h_\theta(x), y) = -y_i log(h_\theta(x)) - (1-y_i) log(1-h_\theta(x))$
    - 模型训练的方式是“joint training”：
        - 同时优化所有参数，在训练阶段就考虑了Wide组件和Deep组件以及他们之间的加权求和的权重参数。
        - Wide组件仅需要补充Deep组件的弱点，通常是一小部分交叉特征的变换，而不是一个整个full-size的Wide模型。
        - paper原文内容：
          In comparison, for joint training the wide part only needs to complement the weaknesses of the deep part with a small number of cross-product feature transformations, rather than a full-size wide model.
    - Wide & Deep Model，整个joint training过程为：
        - 宏观过程是基于BP算法来实现优化过程；
        - Wide部分，默认是使用带L1正则项的ftrl进行求解；
        - Deep部分，默认是使用AdaGrad Optimizer进行求解；
        - PS：**这里的**optimizer**最好自己指定。**

3、 TensorFlow实现

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import tempfile
import pandas as pd
import urllib
import numpy as np
import warnings

from __future__ import print_function

warnings.filterwarnings("ignore")

# Categorical base columns.
gender = tf.contrib.layers.sparse_column_with_keys(column_name="gender", keys=["Female",
"Male"])
race = tf.contrib.layers.sparse_column_with_keys(column_name="race", keys=["Amer-Indian-
Eskimo", "Asian-Pac-Islander", "Black", "Other", "White"])
education = tf.contrib.layers.sparse_column_with_hash_bucket("education",
hash_bucket_size=1000)
relationship = tf.contrib.layers.sparse_column_with_hash_bucket("relationship",
hash_bucket_size=100)
workclass = tf.contrib.layers.sparse_column_with_hash_bucket("workclass",
hash_bucket_size=100)
occupation = tf.contrib.layers.sparse_column_with_hash_bucket("occupation",
hash_bucket_size=1000)
native_country = tf.contrib.layers.sparse_column_with_hash_bucket("native_country",
hash_bucket_size=1000)

# Continuous base columns.
age = tf.contrib.layers.real_valued_column("age")
age_buckets = tf.contrib.layers.bucketized_column(age, boundaries=[18, 25, 30, 35, 40, 45,
50, 55, 60, 65])
education_num = tf.contrib.layers.real_valued_column("education_num")
capital_gain = tf.contrib.layers.real_valued_column("capital_gain")
capital_loss = tf.contrib.layers.real_valued_column("capital_loss")
hours_per_week = tf.contrib.layers.real_valued_column("hours_per_week")

# wide 组件的特征部分，可以看出cross-product transformation是需要algo engineer指定的。
# hash_bucket_size 参数指定 hash bucket 的桶个数，特征交叉的组合个数越多，hash_bucket_size 也应
相应增加，从而减小哈希冲突
wide_columns = [
  gender, native_country, education, occupation, workclass, relationship, age_buckets,
  tf.contrib.layers.crossed_column([education, occupation], hash_bucket_size=int(1e4)),
  tf.contrib.layers.crossed_column([native_country, occupation], hash_bucket_size=int(1e4)),
  tf.contrib.layers.crossed_column([age_buckets, education, occupation],
hash_bucket_size=int(1e6))]

# deep 组件中，对离散型的特征进行embedding，并指定维度（即latent factor的维度）
deep_columns = [
  tf.contrib.layers.embedding_column(workclass, dimension=8),
  tf.contrib.layers.embedding_column(education, dimension=8),
  tf.contrib.layers.embedding_column(gender, dimension=8),
  tf.contrib.layers.embedding_column(relationship, dimension=8),
  tf.contrib.layers.embedding_column(native_country, dimension=8),
  tf.contrib.layers.embedding_column(occupation, dimension=8),
  age, education_num, capital_gain, capital_loss, hours_per_week]

model_dir = tempfile.mkdtemp()
m = tf.contrib.learn.DNNLinearCombinedClassifier(
    dnn_optimizer=Adagrad,  # default值为Adagrad
    model_dir=model_dir,
```

```
    linear_feature_columns=wide_columns,
    dnn_feature_columns=deep_columns,
    dnn_hidden_units=[100, 50])  # 定义两个hidden layer，神经元的个数分别为100和50

# Define the column names for the data sets.
COLUMNS = ["age", "workclass", "fnlwgt", "education", "education_num",
  "marital_status", "occupation", "relationship", "race", "gender",
  "capital_gain", "capital_loss", "hours_per_week", "native_country", "income_bracket"]
LABEL_COLUMN = 'label'
CATEGORICAL_COLUMNS = ["workclass", "education", "marital_status", "occupation",
                       "relationship", "race", "gender", "native_country"]
CONTINUOUS_COLUMNS = ["age", "education_num", "capital_gain", "capital_loss",
                      "hours_per_week"]

# Download the training and test data to temporary files.
# Alternatively, you can download them yourself and change train_file and
# test_file to your own paths.
train_file = tempfile.NamedTemporaryFile()
test_file = tempfile.NamedTemporaryFile()
urllib.urlretrieve("http://mlr.cs.umass.edu/ml/machine-learning-databases/adult/adult.data",
train_file.name)
urllib.urlretrieve("http://mlr.cs.umass.edu/ml/machine-learning-databases/adult/adult.test",
test_file.name)

# Read the training and test data sets into Pandas dataframe.
df_train = pd.read_csv(train_file, names=COLUMNS, skipinitialspace=True)
df_test = pd.read_csv(test_file, names=COLUMNS, skipinitialspace=True, skiprows=1)
df_train[LABEL_COLUMN] = (df_train['income_bracket'].apply(lambda x: '>50K' in
x)).astype(int)
df_test[LABEL_COLUMN] = (df_test['income_bracket'].apply(lambda x: '>50K' in x)).astype(int)

def input_fn(df):
  # Creates a dictionary mapping from each continuous feature column name (k) to
  # the values of that column stored in a constant Tensor.
  continuous_cols = {k: tf.constant(df[k].values)
                     for k in CONTINUOUS_COLUMNS}
  # Creates a dictionary mapping from each categorical feature column name (k)
  # to the values of that column stored in a tf.SparseTensor.
  categorical_cols = {k: tf.SparseTensor(
      indices=[[i, 0] for i in range(df[k].size)],
      values=df[k].values,
      dense_shape=[df[k].size, 1])
                      for k in CATEGORICAL_COLUMNS}
  # Merges the two dictionaries into one.
  feature_cols = dict(continuous_cols.items() + categorical_cols.items())
  # Converts the label column into a constant Tensor.
  label = tf.constant(df[LABEL_COLUMN].values)
  # Returns the feature columns and the label.
  return feature_cols, label

def train_input_fn():
  return input_fn(df_train)

def eval_input_fn():
  return input_fn(df_test)

print('df_train shape:',np.array(df_train).shape)
print('df_test shape:',np.array(df_test).shape)

m.fit(input_fn=train_input_fn, steps=200)
results = m.evaluate(input_fn=eval_input_fn, steps=1)
for key in sorted(results):
    print("%s: %s" % (key, results[key]))
```