推荐系统系列（一）：FM理论与实践

默存 SOTA Lab 2019-11-08

背景

在推荐领域CTR（click-through rate）预估任务中，最常用到的baseline模型就是LR（Logistic Regression）。对数据进行特征工程，构造出大量单特征，编码之后送入模型。这种线性模型的优势在于，运算速度快可解释性强，在特征挖掘完备且训练数据充分的前提下能够达到一定精度。但这种模型的缺点也是较为明显的：

1. 模型并未考虑到特征之间的关系 $y = w_0 + \sum_{i=1}^{n} w_i x_i$ 。在实践经验中，对特征进行交叉组合往往能够更好地提升模型效果。
2. 对于多取值的categorical特征进行one-hot编码，具有高度稀疏性，带来维度灾难问题。

FM（Factorization Machine）模型就是针对在特征组合过程中遇到的上述问题而提出的一种高效的解决方案[1]。由于FM优越的性能表现，后续出现了一系列FM变种模型，从浅层模型到深度推荐模型中都有FM的影子。

分析

1. FM定义

FM以特征组合进行切入点，在公式定义中引入特征交叉项，弥补了一般线性模型未考虑特征间关系的缺憾。公式如下（FM模型可拓展到高阶，但为简化且不失一般性，这里只讨论二阶交叉）[1]：

$$y = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} x_i x_j \qquad (1)$$

与一般线性模型相比，公式（1）仅多了一个二阶交叉项，模型参数多了 $\frac{n(n+1)}{2}$ 个。虽然这种显式交叉的方式能够刻画特征间关系，但是对公式求解带来困难。

因为大量特征进行one-hot表示之后具有高度稀疏性的问题，所以公式（1）中的 $x_i x_j$ 同样会产生大量的0值。参数学习不充分，直接导致 w_{ij} 无法通过训练得到。（解释：令 $x_i x_j = X$，则 $\frac{\partial y}{\partial w_{ij}} = X$，又因 $X = 0$，所以 $w_{ij}^{new} = w_{ij}^{old} + \alpha X = w_{ij}^{old}$ ，梯度为0参数无法更新。）

导致这种情况出现的根源在于：特征过于稀疏。我们期望的是找到一种方法，使得 w_{ij} 的求解不受特征稀疏性的影响。

2. 公式改写

为了克服上述困难，需要对FM公式进行改写，使得求解更加顺利。受 **矩阵分解** 的启发，对于每一个特征 x_i 引入辅助向量（隐向量）$V_i = (v_{i1}, v_{i2}, \cdots, v_{ik})$，然后利用$V_iV_j^T$ 对w_{ij} 进行求解。即，做如下假设： $w_{ij} \approx V_iV_j^T$ 。

引入隐向量的好处是：

1. 二阶项的参数量由原来的 $\frac{n(n-1)}{2}$ 降为 kn 。
2. 原先参数之间并无关联关系，但是现在通过隐向量可以建立关系。如，之前 w_{ij} 与 w_{ik} 无关，但是现在 $w_{ij} = \langle V_i, V_j \rangle, w_{ik} = \langle V_i, V_k \rangle$ 两者有共同的 V_i ，也就是说，所有包含 x_ix_j 的非零组合特征（存在某个 $j \neq i$ ，使得 $x_ix_j \neq 0$ ）的样本都可以用来学习隐向量 V_i ，这很大程度上避免了数据稀疏性造成的影响。[2]

现在可以将公式（1）进行改写：

$$y = w_0 + \sum_{i=1}^{n} w_ix_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \langle V_i, V_j \rangle x_ix_j \qquad (2)$$

重心转移到如何求解公式（2）后面的二阶项。

预备知识：

首先了解 *对称* 矩阵上三角求和，设矩阵为 M：

$$M = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{pmatrix}_{n*n}$$

其中，$m_{ij} = m_{ji}$ 。

令上三角元素和为 A ，即 $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij} = A$ 。那么，M 的所有元素之和等于 $2 * A + tr(M)$ ，$tr(M)$为矩阵的迹。

$$\sum_{i=1}^{n} \sum_{j=1}^{n} m_{ij} = 2 * \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij} + \sum_{i=1}^{n} m_{ii}$$

可得，

$$A=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}m_{ij}=\frac{1}{2}*\left\{\sum_{i=1}^{n}\sum_{j=1}^{n}m_{ij}-\sum_{i=1}^{n}m_{ii}\right\}$$

正式改写：

有了上述预备知识，可以对公式（2）的二阶项进行推导：

$$\begin{aligned}
&\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\langle V_i,V_j\rangle x_ix_j \\
&=\frac{1}{2}*\left\{\sum_{i=1}^{n}\sum_{j=i}^{n}\langle V_i,V_j\rangle x_ix_j-\sum_{i=1}^{n}\langle V_i,V_i\rangle x_ix_i\right\} \\
&=\frac{1}{2}*\left\{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k}v_{if}v_{jf}x_ix_j-\sum_{i=1}^{n}\sum_{f=1}^{k}v_{if}v_{if}x_ix_i\right\} \\
&=\frac{1}{2}*\sum_{f=1}^{k}\left\{\sum_{i=1}^{n}\sum_{j=1}^{n}v_{if}x_iv_{jf}x_j-\sum_{i=1}^{n}v_{if}^2x_i^2\right\} \\
&=\frac{1}{2}*\sum_{f=1}^{k}\left\{\left(\sum_{i=1}^{n}v_{if}x_i\right)\left(\sum_{j=1}^{n}v_{jf}x_j\right)-\sum_{i=1}^{n}v_{if}^2x_i^2\right\} \\
&=\frac{1}{2}*\sum_{f=1}^{k}\left\{\left(\sum_{i=1}^{n}v_{if}x_i\right)^2-\sum_{i=1}^{n}v_{if}^2x_i^2\right\}
\end{aligned}\tag{3}$$

结合（2）（3），可以得到：

$$\begin{aligned}
y&=w_0+\sum_{i=1}^{n}w_ix_i+\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\langle V_i,V_j\rangle x_ix_j \\
&=w_0+\sum_{i=1}^{n}w_ix_i+\frac{1}{2}*\sum_{f=1}^{k}\left\{\left(\sum_{i=1}^{n}v_{if}x_i\right)^2-\sum_{i=1}^{n}v_{if}^2x_i^2\right\}
\end{aligned}\tag{4}$$

至此，我们得到了想要的模型表达式。

为什么要将公式（2）改写为公式（4），是因为在改写之前，计算 y 的复杂度为 $O(kn^2)$，改写后的计算复杂度为 $O(kn)$，提高模型推断速度。

3. FM求解

到目前为止已经得到了FM的模型表示（4），如何对模型参数求解呢？可以使用常见的梯度下降法对参数进行求解，为了对参数进行梯度下降更新，需要计算模型各参数的梯度表达式：

当参数为 w_0 时，$\frac{\partial y}{\partial w_0}=1$ 。

当参数为 w_i 时，$\frac{\partial y}{\partial w_i}=x_i$ 。

当参数为 v_{if} 时，只需要关注模型高阶项，当计算参数 v_{if} 的梯度时，其余无关参数可看做常数。

$$\begin{aligned}\frac{\partial y}{\partial v_{if}} &= \partial \frac{1}{2}\left\{\left(\sum_{i=1}^{n} v_{if}x_i\right)^2 - \sum_{i=1}^{n} v_{if}^2 x_i^2\right\} / \partial v_{if} \\ &= \frac{1}{2} * \left\{\frac{\partial\left\{\sum_{i=1}^{n} v_{if}x_i\right\}^2}{\partial v_{if}} - \frac{\partial\left\{\sum_{i=1}^{n} v_{if}^2 x_i^2\right\}}{\partial v_{if}}\right\}\end{aligned} \tag{5}$$

其中：

$$\frac{\partial\left\{\sum_{i=1}^{n} v_{if}^2 x_i^2\right\}}{\partial v_{if}} = 2x_i^2 v_{if} \tag{6}$$

令 $\lambda = \sum_{i=1}^{n} v_{if}x_i$ ，则：

$$\begin{aligned}\frac{\partial\left\{\sum_{i=1}^{n} v_{if}x_i\right\}^2}{\partial v_{if}} &= \frac{\partial \lambda^2}{\partial v_{if}} \\ &= \frac{\partial \lambda^2}{\partial \lambda}\frac{\partial \lambda}{\partial v_{if}} \\ &= 2\lambda * \frac{\partial \sum_{i=1}^{n} v_{if}x_i}{\partial v_{if}} \\ &= 2\lambda * x_i \\ &= 2 * x_i * \sum_{j=1}^{n} v_{jf}x_j\end{aligned} \tag{7}$$

结合公式（5~7），可得：

$$\frac{\partial y}{\partial v_{if}} = x_i \sum_{j=1}^{n} v_{jf}x_j - x_i^2 v_{if} \tag{8}$$

综上，最终模型各参数的梯度表达式如下：

$$\frac{\partial y}{\partial \theta} = \begin{cases} 1, & \text{if } \theta \text{ is } w_0; \\ x_i, & \text{if } \theta \text{ is } w_i; \\ x_i \sum_{j=1}^{n} v_{jf}x_j - x_i^2 v_{if}, & \text{if } \theta \text{ is } v_{if}. \end{cases}$$

4. 性能分析

由第2小节可知，FM进行推断的时间复杂度为 $O(kn)$ 。分析训练的复杂度，依据参数的梯度表达式，$\sum_{j=1}^{n} v_{jf}x_j$ 与 i 无关，在参数更新时可以首先将所有的 $\sum_{j=1}^{n} v_{jf}x_j$ 计算出来，复杂度为 $O(kn)$ ，后续更新所有参数的时间复杂度均为 $O(1)$ ，参数量为 $1+n+kn$ ，所以最终训练的时间复杂度同样为 $O(kn)$ ，其中 n 为特征数，k 为隐向量维数。

FM训练与预测的时间复杂度均为 $O(kn)$ ，是一种十分高效的模型。

5. 优缺点

优点 [1]：

> In total, the advantages of our proposed FM are:
>
> 1. FMs allow parameter estimation under very sparse data where SVMs fail.
> 2. FMs have linear complexity, can be optimized in the primal and do not rely on support vectors like SVMs. We show that FMs scale to large datasets like Netflix with 100 millions of training instances.
> 3. FMs are a general predictor that can work with any real valued feature vector. In contrast to this, other state-of- the-art factorization models work only on very restricted input data. We will show that just by defining the feature vectors of the input data, FMs can mimic state-of-the-art models like biased MF, SVD++, PITF or FPMC.

缺点：

1. 每个特征只引入了一个隐向量，不同类型特征之间交叉没有区分性。FFM模型正是以这一点作为切入进行改进。

实验

FM既可以应用在回归任务，也可以应用在分类任务中。如，在二分类任务中只需在公式（2）最外层套上 $sigmoid$ 函数即可，上述解析都是基于回归任务来进行推导的。

关于模型最终的损失函数同样可以有多种形式，如回归任务可以使用 MSE ，分类任务可以使用 $CrossEntropy$ 等。

1. 代码演示

虽然知道可以通过引入辅助向量进行计算，但是辅助向量是如何与特征 x_i 建立联系的，换句话说，如何通过 x_i 得到辅助向量 V_i ？在使用神经网络实现FM的过程中，将 x_i 的 $embedding$ 作为辅助向量，最终得到的 $embedding$ 向量组也可以看作是对应特征的低维稠密表征，可以应用到其他下游任务中。

1.1 回归任务

本文使用了 $MovieLens100K Dataset$ [3] 作为实验输入，特征组分别为用户编号、电影编号，用户对电影的历史评分作为 $Label$ 。

具体代码实现如下：

```
# -*- coding:utf-8 -*-
import pandas as pd
import numpy as np
from scipy.sparse import csr
from itertools import count
from collections import defaultdict
import tensorflow as tf


def vectorize_dic(dic, label2index=None, hold_num=None):

    if label2index == None:
        d = count(0)
        label2index = defaultdict(lambda: next(d))  # 数值映射表

    sample_num = len(list(dic.values())[0])  # 样本数
    feat_num = len(list(dic.keys()))  # 特征数
    total_value_num = sample_num * feat_num

    col_ix = np.empty(total_value_num, dtype=int)

    i = 0
    for k, lis in dic.items():
        col_ix[i::feat_num] = [label2index[str(k) + str(el)] for el in lis]
        i += 1

    row_ix = np.repeat(np.arange(sample_num), feat_num)
    data = np.ones(total_value_num)

    if hold_num isNone:
        hold_num = len(label2index)

    left_data_index = np.where(col_ix < hold_num)  # 为了剔除不在train set中出现的test set数据

    return csr.csr_matrix(
        (data[left_data_index], (row_ix[left_data_index], col_ix[left_data_index])),
        shape=(sample_num, hold_num)), label2index
```

```
def batcher(X_, y_, batch_size=-1):

    assert X_.shape[0] == len(y_)

    n_samples = X_.shape[0]
    if batch_size == -1:
        batch_size = n_samples
    if batch_size < 1:
        raise ValueError('Parameter batch_size={} is unsupported'.format(batch_size))

    for i in range(0, n_samples, batch_size):
        upper_bound = min(i + batch_size, n_samples)
        ret_x = X_[i:upper_bound]
        ret_y = y_[i:upper_bound]
        yield(ret_x, ret_y)

def load_dataset():
    cols = ['user', 'item', 'rating', 'timestamp']
    train = pd.read_csv('data/ua.base', delimiter='\t', names=cols)
    test = pd.read_csv('data/ua.test', delimiter='\t', names=cols)

    x_train, label2index = vectorize_dic({'users': train.user.values, 'items': train.item.values})
    x_test, label2index = vectorize_dic({'users': test.user.values, 'items': test.item.values}, lab

    y_train = train.rating.values
    y_test = test.rating.values

    x_train = x_train.todense()
    x_test = x_test.todense()

    return x_train, x_test, y_train, y_test

x_train, x_test, y_train, y_test = load_dataset()

print("x_train shape: ", x_train.shape)
print("x_test shape: ", x_test.shape)
print("y_train shape: ", y_train.shape)
print("y_test shape: ", y_test.shape)

vec_dim = 10
batch_size = 1000
epochs = 10
learning_rate = 0.001
sample_num, feat_num = x_train.shape

x = tf.placeholder(tf.float32, shape=[None, feat_num], name="input_x")
```

```
y = tf.placeholder(tf.float32, shape=[None,1], name="ground_truth")

w0 = tf.get_variable(name="bias", shape=(1), dtype=tf.float32)
W = tf.get_variable(name="linear_w", shape=(feat_num), dtype=tf.float32)
V = tf.get_variable(name="interaction_w", shape=(feat_num, vec_dim), dtype=tf.float32)

linear_part = w0 + tf.reduce_sum(tf.multiply(x, W), axis=1, keep_dims=True)
interaction_part = 0.5 * tf.reduce_sum(tf.square(tf.matmul(x, V)) - tf.matmul(tf.square(x), tf.squa
y_hat = linear_part + interaction_part
loss = tf.reduce_mean(tf.square(y - y_hat))
train_op = tf.train.AdamOptimizer(learning_rate).minimize(loss)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for e in range(epochs):
        step = 0
        print("epoch:{}".format(e))
        for batch_x, batch_y in batcher(x_train, y_train, batch_size):
            sess.run(train_op, feed_dict={x:batch_x, y:batch_y.reshape(-1, 1)})
            step += 1
            if step % 10 == 0:
                for val_x, val_y in batcher(x_test, y_test):
                    train_loss = sess.run(loss, feed_dict={x:batch_x, y:batch_y.reshape(-1, 1)})
                    val_loss = sess.run(loss, feed_dict={x:val_x, y:val_y.reshape(-1, 1)})
                    print("batch train_mse={}, val_mse={}".format(train_loss, val_loss))

    for val_x, val_y in batcher(x_test, y_test):
        val_loss = sess.run(loss, feed_dict={x: val_x, y: val_y.reshape(-1, 1)})
        print("test set rmse = {}".format(np.sqrt(val_loss)))
```

实验结果：

```
epoch:0
batch train_mse=19.54930305480957, val_mse=19.687997817993164
batch train_mse=16.957233428955078, val_mse=19.531404495239258
batch train_mse=18.544944763183594, val_mse=19.376962661743164
batch train_mse=18.870519638061523, val_mse=19.222412109375
batch train_mse=18.769777297973633, val_mse=19.070764541625977
batch train_mse=19.383392333984375, val_mse=18.915040969848633
batch train_mse=17.26403045654297, val_mse=18.75937843322754
batch train_mse=17.652183532714844, val_mse=18.6033935546875
batch train_mse=18.331804275512695, val_mse=18.447608947753906
......
epoch:9
```

```
batch train_mse=1.394300103187561, val_mse=1.4516444206237793
batch train_mse=1.2031371593475342, val_mse=1.4285767078399658
batch train_mse=1.1761484146118164, val_mse=1.4077649116516113
batch train_mse=1.134848952293396, val_mse=1.3872103691101074
batch train_mse=1.2191411256790161, val_mse=1.3692644834518433
batch train_mse=1.572729468345642, val_mse=1.3509554862976074
batch train_mse=1.3323310613632202, val_mse=1.3339732885360718
batch train_mse=1.1601723432540894, val_mse=1.3183823823928833
batch train_mse=1.2751621007919312, val_mse=1.3023829460144043
test set rmse = 1.1405380964279175
```

1.2 分类任务

使用更全的 $MovieLens100KDataset$ 特征，将评分大于3分的样本作为正类，其他为负类，构造二分类任务。核心代码如下

```
class FM(object):
    def __init__(self, vec_dim, feat_num, lr, lamda):
        self.vec_dim = vec_dim
        self.feat_num = feat_num
        self.lr = lr
        self.lamda = lamda

        self._build_graph()

    def _build_graph(self):
        self.add_input()
        self.inference()

    def add_input(self):
        self.x = tf.placeholder(tf.float32, shape=[None, self.feat_num], name='input_x')
        self.y = tf.placeholder(tf.float32, shape=[None], name='input_y')

    def inference(self):
        with tf.variable_scope('linear_part'):
            w0 = tf.get_variable(name='bias', shape=[1], dtype=tf.float32)
            self.W = tf.get_variable(name='linear_w', shape=[self.feat_num], dtype=tf.float32)
            self.linear_part = w0 + tf.reduce_sum(tf.multiply(self.x, self.W), axis=1)
        with tf.variable_scope('interaction_part'):
            self.V = tf.get_variable(name='interaction_w', shape=[self.feat_num, self.vec_dim], dty
            self.interaction_part = 0.5 * tf.reduce_sum(
                tf.square(tf.matmul(self.x, self.V)) - tf.matmul(tf.square(self.x), tf.square(self.
                axis=1
            )
        self.y_logits = self.linear_part + self.interaction_part
        self.y_hat = tf.nn.sigmoid(self.y_logits)
```

```
        self.pred_label = tf.cast(self.y_hat > 0.5, tf.int32)
        self.loss = -tf.reduce_mean(self.y*tf.log(self.y_hat+1e-8) + (1-self.y)*tf.log(1-self.y_hat
        self.reg_loss = self.lamda*(tf.reduce_mean(tf.nn.l2_loss(self.W)) + tf.reduce_mean(tf.nn.l2
        self.total_loss = self.loss + self.reg_loss

        self.train_op = tf.train.AdamOptimizer(self.lr).minimize(self.total_loss)
```

实验结果：

```
Iter:  59400, Train acc: 0.7812, Val acc: 0.6867, Val auc: 0.7285, Val loss: 0.614005, Flag:
Iter:  59600, Train acc: 0.8125, Val acc:  0.684, Val auc: 0.7294, Val loss: 0.615628, Flag: *
Iter:  59800, Train acc:  0.875, Val acc: 0.6665, Val auc: 0.7282, Val loss: 0.625017, Flag:
Iter:  60000, Train acc: 0.9375, Val acc: 0.6767, Val auc: 0.7282, Val loss: 0.617686, Flag:
Iter:  60200, Train acc:   0.75, Val acc: 0.6815, Val auc: 0.7277, Val loss: 0.614763, Flag:
Iter:  60400, Train acc: 0.9062, Val acc:  0.681, Val auc: 0.7283, Val loss: 0.614414, Flag:
Iter:  60600, Train acc: 0.6875, Val acc: 0.6853, Val auc: 0.7291, Val loss: 0.621548, Flag:
Iter:  60800, Train acc:  0.625, Val acc:  0.679, Val auc: 0.7288, Val loss: 0.617327, Flag:
Iter:  61000, Train acc: 0.7812, Val acc: 0.6835, Val auc: 0.7293, Val loss: 0.616952, Flag:
Iter:  61200, Train acc: 0.8125, Val acc:  0.686, Val auc: 0.7292, Val loss: 0.614379, Flag:
Iter:  61400, Train acc: 0.6562, Val acc:  0.688, Val auc: 0.7284, Val loss: 0.613859, Flag:
Iter:  61600, Train acc: 0.6875, Val acc: 0.6725, Val auc: 0.7279, Val loss: 0.618824, Flag:
No optimization for a long time, auto-stopping...
====== let's test =====
Test acc: 0.6833, Test auc: 0.7369
```

2. 注意事项

- 虽然FM可以应用于任意数值类型的数据上，但是需要注意对输入特征数值进行预处理。优先进行特征归一化，其次再进行样本归一化。[2]
- FM不仅可以用于rank阶段，同时可以用于向量召回。[8]

reference

- [1] Rendle, S. (2010, December). Factorization machines. In 2010 IEEE International Conference on Data Mining (pp. 995-1000). IEEE.
- [2] https://tech.meituan.com/2016/03/03/deep-understanding-of-ffm-principles-and-practices.html
- [3] https://grouplens.org/datasets/movielens/100k/
- [4] https://www.jianshu.com/p/152ae633fb00
- [5] http://nowave.it/factorization-machines-with-tensorflow.html