FFM模型理论和实践

1、FFM理论

在CTR预估中，经常会遇到one-hot类型的变量，one-hot类型变量会导致严重的数据特征稀疏的情况，为了解决这一问题，在上一讲中，我们介绍了FM算法。这一讲我们介绍一种在FM基础上发展出来的算法-FFM（Field-aware Factorization Machine）。

FFM模型中引入了类别的概念，即field。还是拿上一讲中的数据来讲，先看下图：

Clicked?	Country	Day	Ad_type
1	USA	26/11/15	Movie
0	China	1/7/14	Game
1	China	19/2/15	Game

在上面的广告点击案例中，

“Day=26/11/15”、“Day=1/7/14”、“Day=19/2/15”这三个特征都是代表日期的，可以放到同一个field中。同理，Country也可以放到一个field中。简单来说，同一个categorical特征经过One-Hot编码生成的数值特征都可以放到同一个field，包括用户国籍，广告类型，日期等等。

在FFM中，每一维特征 x_i，针对其它特征的每一种field f_j，都会学习一个隐向量 v_{i},f_j。因此，隐向量不仅与特征相关，也与field相关。也就是说，“Day=26/11/15”这个特征与“Country”特征和“Ad_type"特征进行关联的时候使用不同的隐向量，这与“Country”和“Ad_type”的内在差异相符，也是FFM中“field-aware”的由来。

假设样本的 n个特征属于 f个field，那么FFM的二次项有 nf个隐向量。而在FM模型中，每一维特征的隐向量只有一个。FM可以看作FFM的特例，是把所有特征都归属到一个field时的FFM模型。根据FFM的field敏感特性，可以导出其模型方程。

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_{i,f_j}, \mathbf{v}_{j,f_i} \rangle x_i x_j$$

可以看到，如果隐向量的长度为 k，那么FFM的二次参数有 nfk 个，远多于FM模型的 nk个。此外，由于隐向量与field相关，FFM二次项并不能够化简，其预测复杂度是 O(kn^2)。

下面以一个例子简单说明FFM的特征组合方式。输入记录如下：

User	Movie	Genre	Price
YuChin	3Idiots	Comedy, Drama	$9.99

这条记录可以编码成5个特征，其中“Genre=Comedy”和“Genre=Drama”属于同一个field，“Price”是数值型，不用One-Hot编码转换。为了方便说明FFM的样本格式，我们将所有的特征和对应的field映射成整数编号。

Field name	Field index	Feature name	Feature index
User	1	User=YuChin	1
Movie	2	Movie=3Idiots	2
Genre	3	Genre=Comedy	3
Price	4	Genre=Drama	4
		Price	5

那么，FFM的组合特征有10项，如下图所示。

$$
\begin{aligned}
&\langle \mathbf{v}_{1,2}, \mathbf{v}_{2,1} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,3}, \mathbf{v}_{3,1} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,3}, \mathbf{v}_{4,1} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,4}, \mathbf{v}_{5,1} \rangle \cdot 1 \cdot 9.99 \\
&+ \langle \mathbf{v}_{2,3}, \mathbf{v}_{3,2} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{2,3}, \mathbf{v}_{4,2} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{2,4}, \mathbf{v}_{5,2} \rangle \cdot 1 \cdot 9.99 \\
&+ \langle \mathbf{v}_{3,3}, \mathbf{v}_{4,3} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{3,4}, \mathbf{v}_{5,3} \rangle \cdot 1 \cdot 9.99 \\
&+ \langle \mathbf{v}_{4,4}, \mathbf{v}_{5,3} \rangle \cdot 1 \cdot 9.99
\end{aligned}
$$

其中，红色是field编号，蓝色是特征编号。

2、FFM实现细节

这里讲得只是一种FFM的实现方式，并不是唯一的。

损失函数

FFM将问题定义为分类问题，使用的是logistic loss，同时加入了正则项

$$\min_{\mathbf{w}} \sum_{i=1}^{L} \log\left(1+\exp\{-y_i\phi(\mathbf{w},\mathbf{x}_i)\}\right)+\frac{\lambda}{2}\|\mathbf{w}\|^2$$

什么，这是logisitc loss？第一眼看到我是懵逼的，逻辑回归的损失函数我很熟悉啊，不是长这样的啊？其实是我目光太短浅了。逻辑回归其实是有两种表述方式的损失函数的，取决于你将类别定义为0和1还是1和-1。大家可以参考下下面的文章：https://www.cnblogs.com/ljygoodgoodstudydaydayup/p/6340129.html。当我们将类别设定为1和-1的时候，逻辑回归的损失函数就是上面的样子。

随机梯度下降

训练FFM使用的是随机梯度下降方法，即每次只选一条数据进行训练，这里还有必要补一补梯度下降的知识，梯度下降是有三种方式的，截图取自参考文献3：

4.1 批量梯度下降法（Batch Gradient Descent）

批量梯度下降法，是梯度下降法最常用的形式，具体做法也就是在更新参数时使用所有的样本来进行更新，这个方法对应于前面3.3.1的线性回归的梯度下降算法，也就是说3.3.1的梯度下降算法就是批量梯度下降法。

$$\theta_i=\theta_i-\alpha\sum_{j=0}^{m}(h_\theta(x_0^{(j)},x_1^{(j)},\ldots x_n^{(j)})-y_j)x_i^{(j)}$$

由于我们有m个样本，这里求梯度的时候就用了所有m个样本的梯度数据。

4.2 随机梯度下降法（Stochastic Gradient Descent）

随机梯度下降法，其实和批量梯度下降法原理类似，区别在与求梯度时没有用所有的m个样本的数据，而是仅仅选取一个样本j来求梯度。对应的更新公式是：

$$\theta_i=\theta_i-\alpha(h_\theta(x_0^{(j)},x_1^{(j)},\ldots x_n^{(j)})-y_j)x_i^{(j)}$$

随机梯度下降法，和4.1的批量梯度下降法是两个极端，一个采用所有数据来梯度下降，一个用一个样本来梯度下降。自然各自的优缺点都非常突出。对于训练速度来说，随机梯度下降法由于每次仅仅采用一个样本来迭代，训练速度很快，而批量梯度下降法在样本量很大的时候，训练速度不能让人满意。对于准确度来说，随机梯度下降法用于仅仅用一个样本决定梯度方向，导致解很有可能不是最优。对于收敛速度来说，由于随机梯度下降法一次迭代一个样本，导致迭代方向变化很大，不能很快的收敛到局部最优解。

那么，有没有一个中庸的办法能够结合两种方法的优点呢？有！这就是4.3的小批量梯度下降法。

4.3 小批量梯度下降法（Mini-batch Gradient Descent）

小批量梯度下降法是批量梯度下降法和随机梯度下降法的折衷，也就是对于m个样本，我们采用x个样子来迭代，1<x<m。一般可以取x=10，当然根据样本的数据，可以调整这个x的值。对应的更新公式是：

$$\theta_i=\theta_i-\alpha\sum_{j=t}^{t+x-1}(h_\theta(x_0^{(j)},x_1^{(j)},\ldots x_n^{(j)})-y_j)x_i^{(j)}$$

总给人一种怪怪的感觉。batch为什么是全量的数据呢，哈哈。

3、tensorflow实现代码

本文代码的github地址：
https://github.com/princewen/tensorflow_practice/tree/master/recommendation-FFM-Demo

这里我们只讲解一些细节，具体的代码大家可以去github上看：

生成数据
这里我没有找到合适的数据，就自己产生了一点数据，数据涉及20维特征，前十维特征是一个field，后十维是一个field:

```
def gen_data():
    labels = [-1,1]
    y = [np.random.choice(labels,1)[0] for _ in range(all_data_size)]
    x_field = [i // 10 for i in range(input_x_size)]
    x = np.random.randint(0,2,size=(all_data_size,input_x_size))
    return x,y,x_field
```

定义权重项
在ffm中，有三个权重项，首先是bias，然后是一维特征的权重，最后是交叉特征的权重：

```
def createTwoDimensionWeight(input_x_size,field_size,vector_dimension):
    weights = tf.truncated_normal([input_x_size,field_size,vector_dimension])

    tf_weights = tf.Variable(weights)

    return tf_weights

def createOneDimensionWeight(input_x_size):
    weights = tf.truncated_normal([input_x_size])
    tf_weights = tf.Variable(weights)
    return tf_weights

def createZeroDimensionWeight():
    weights = tf.truncated_normal([1])
    tf_weights = tf.Variable(weights)
    return tf_weights
```

计算估计值
估计值的计算这里不能项FM一样先将公式化简再来做，对于交叉特征，只能写两重循环，所以对于特别多的特征的情况下，真的计算要爆炸呀！

```
def inference(input_x,input_x_field,zeroWeights,oneDimWeights,thirdWeight):
    """计算回归模型输出的值"""

    secondValue = tf.reduce_sum(tf.multiply(oneDimWeights,input_x,name='secondValue'))

    firstTwoValue = tf.add(zeroWeights, secondValue, name="firstTwoValue")

    thirdValue = tf.Variable(0.0,dtype=tf.float32)
    input_shape = input_x_size
```

```
    for i in range(input_shape):
        featureIndex1 = I
        fieldIndex1 = int(input_x_field[I])
        for j in range(i+1,input_shape):
            featureIndex2 = j
            fieldIndex2 = int(input_x_field[j])
            vectorLeft = tf.convert_to_tensor([[featureIndex1,fieldIndex2,i] for i in 
range(vector_dimension)])
            weightLeft = tf.gather_nd(thirdWeight,vectorLeft)
            weightLeftAfterCut = tf.squeeze(weightLeft)

            vectorRight = tf.convert_to_tensor([[featureIndex2,fieldIndex1,i] for i in 
range(vector_dimension)])
            weightRight = tf.gather_nd(thirdWeight,vectorRight)
            weightRightAfterCut = tf.squeeze(weightRight)

            tempValue = tf.reduce_sum(tf.multiply(weightLeftAfterCut,weightRightAfterCut))

            indices2 = [I]
            indices3 = [j]

            xi = tf.squeeze(tf.gather_nd(input_x, indices2))
            xj = tf.squeeze(tf.gather_nd(input_x, indices3))

            product = tf.reduce_sum(tf.multiply(xi, xj))

            secondItemVal = tf.multiply(tempValue, product)

            tf.assign(thirdValue, tf.add(thirdValue, secondItemVal))

    return tf.add(firstTwoValue,thirdValue)
```

定义损失函数

损失函数我们就用逻辑回归损失函数来算，同时加入正则项：

```
lambda_w = tf.constant(0.001, name='lambda_w')
lambda_v = tf.constant(0.001, name='lambda_v')

zeroWeights = createZeroDimensionWeight()

oneDimWeights = createOneDimensionWeight(input_x_size)

thirdWeight = createTwoDimensionWeight(input_x_size,  # 创建二次项的权重变量
                                       field_size,
                                       vector_dimension)  # n * f * k

y_ = inference(input_x, trainx_field,zeroWeights,oneDimWeights,thirdWeight)

l2_norm = tf.reduce_sum(
    tf.add(
        tf.multiply(lambda_w, tf.pow(oneDimWeights, 2)),
        tf.reduce_sum(tf.multiply(lambda_v, tf.pow(thirdWeight, 2)),axis=[1,2])
    )
)

loss = tf.log(1 + tf.exp(input_y * y_)) + l2_norm

train_step = tf.train.GradientDescentOptimizer(learning_rate=lr).minimize(loss)
```

训练

接下来就是训练了，每次只用喂一个数据就好：

```
input_x_batch = trainx[t]
input_y_batch = trainy[t]
predict_loss,_, steps = sess.run([loss,train_step, global_step],
                          feed_dict={input_x: input_x_batch, input_y: input_y_batch})
```

跑的是相当的慢，我们来看看效果吧：

```
After  0 training   step(s)   ,   loss    on    training    batch   is  [ 1.34960818]
[0 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0] 1
After  0 training   step(s)   ,   loss    on    training    batch   is  [ 1.03773224]
[0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 1] 1
After  0 training   step(s)   ,   loss    on    training    batch   is  [ 0.14303313]
```


参考文章

1、

https://tech.meituan.com/deep-understanding-of-ffm-principles-and-practices.html

2、https://www.cnblogs.com/ljygoodgoodstudydaydayup/p/6340129.html

3、https://www.cnblogs.com/pinard/p/5970503.html

石晓文，中国人民大学信息学院在读研究生，美团外卖算法实习生

天善社区：

https://www.hellobi.com/u/58654/articles