CTR模型系列:FM， FFM和AFM

原创 kylekzhang 算法工程师养成之路 2019-06-26

导引

因子分解机(FM)作为第一个对LR引入二阶特征对模型，引发了类似的很多模型，包括Filed-aware FM和Attention FM。之所以选择这三个放在一起说，是因为他们有共同指出。本节对这三个模型做一个简单的介绍。
本文所有代码均可以参考：
https://github.com/End-the-cold-night/deeprec/tree/master/deeprec/ranking/ctr
deeprec是一个通用的推荐和广告算法库，目前支持常见的CTR模型，后续会陆续补充CVR，Graph embedding和kg相关的算法。欢迎大家star和contribute。

FM模型

01

我们知道，对于稀疏数据，独立编码会导致维度很大，假设独热编码展开后全部特征长度为n，则下面式中：

$$\phi_{\mathrm{FM}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (\boldsymbol{w}_{j_1} \cdot \boldsymbol{w}_{j_2}) x_{j_1} x_{j_2}.$$

共计有n*(n-1)/2个参数，这么大的参数需要对应的x都为1，因此会导致训练不充分。FM的思想是借鉴了矩阵分解的思想，即借助W矩阵是对称矩阵，因此有：

$$\hat{\mathbf{W}} = \mathbf{V}\mathbf{V}^T = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{pmatrix} \begin{pmatrix} \mathbf{v}_1^T & \mathbf{v}_2^T & \cdots & \mathbf{v}_n^T \end{pmatrix}$$

其中V就是embedding矩阵，FM简化如下：

$$
\begin{aligned}
&\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \\
&= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j - \frac{1}{2} \sum_{i=1}^{n} \langle \mathbf{v}_i, \mathbf{v}_i \rangle x_i x_i \\
&= \frac{1}{2} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{f=1}^{k} v_{i,f} v_{j,f} x_i x_j - \sum_{i=1}^{n} \sum_{f=1}^{k} v_{i,f} v_{i,f} x_i x_i \right) \\
&= \frac{1}{2} \sum_{f=1}^{k} \left(\left(\sum_{i=1}^{n} v_{i,f} x_i \right) \left(\sum_{j=1}^{n} v_{j,f} x_j \right) - \sum_{i=1}^{n} v_{i,f}^2 x_i^2 \right) \\
&= \frac{1}{2} \sum_{f=1}^{k} \left(\left(\sum_{i=1}^{n} v_{i,f} x_i \right)^2 - \sum_{i=1}^{n} v_{i,f}^2 x_i^2 \right)
\end{aligned}
$$

代码实现：

```python
def fm(embeddings):
    # input: embeddings should be,  bs * fs * es,
    # output: fm,
    summed_features_emb = tf.reduce_sum(embeddings, 1)  # bs * es
    summed_features_emb_square = tf.square(summed_features_emb)  # bs * es
    squared_features_emb = tf.square(embeddings)
    squared_sum_features_emb = tf.reduce_sum(squared_features_emb, 1)  # bs * es
    y_second_order = 0.5 * tf.subtract(summed_features_emb_square, squared_sum_features_emb)  # bs * es
    return y_second_order
```

完整的代码实现，需要加入一阶的部分，即：

```python
sprase_feature, self.sprase_data_linear_embedding = \
    get_linear_embedding(self.feature_config_dict,self.sprase_data, self.number_of_sprase_feature)
```

对于序列数据，FM是直接求平均，即：

```python
# sequence data
if self.number_of_sequence_feature:
    self.sequence_data_embedding= get_sequence_embedding(
        self.embedding_dict, self.masked_sequence_data, self.sequence_feature_name, embedding_size)
    # FM use the average of the embedding directly.
    self.sequence_data_embedding = tf.reduce_mean(self.sequence_data_embedding, axis=1)  # bs * es
    out = tf.concat([out, self.sequence_data_embedding, fm_out], axis=1)
```

将以上一阶、二阶和稀疏数据特征concat到一起，就可以进过一个softmax获得输出：

```python
self.logits = tf.layers.dense(out, 1, activation=None,
                              kernel_regularizer=tf.contrib.layers.l2_regularizer(l2_reg_linear))
```

完整代码参考我的github，
https://github.com/End-the-cold-night/deeprec/blob/master/deeprec/ranking/ctr/model/Fm.py

调包使用：
 from deeprec.ranking.ctr import FM

02 FFM

Filed-aware FM

FM 假设特征i在和其它特征做组合时候保持同一个表示，会对不同组合空间的学习带来一定的难度。FFM为了解决这个，思路很简单粗暴，即让每个特征和不同特征组合时候有不同的表示。
直接看公式，比较清晰，即

$$\phi_{\mathrm{FFM}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (\boldsymbol{w}_{j_1,f_2} \cdot \boldsymbol{w}_{j_2,f_1}) x_{j_1} x_{j_2},$$

对比FM的第一个公式，可以看出FFM变量的下标多了filed维度。其带来的第一个问题就是参数数量的显著增加，相对FM的参数n*k，扩大到了n*f*k，但是文章作者提到，因为FFM相对FM需要去学习简单区域的特征，因此k可以很小，即

$$k_{\mathrm{FFM}} \ll k_{\mathrm{FM}},$$

但是主要的问题实际在计算上，FFM 的实现不能直接化简，需要把输入换成独热编码的，即：

```
sprase_data_list = tf.split(self.sprase_data, self.number_of_sprase_feature, axis=1)
sprase_data_embedding_list = []
temp = 0
for var in sprase_feature:
    sprase_data_embedding_list.append(tf.one_hot(sprase_data_list[temp], self.feature_config_dict[var]))
    temp += 1
self.inputs = tf.concat(sprase_data_embedding_list, axis=2)  # bs * feature_size
self.inputs = tf.squeeze(self.inputs, 1)
```


之后根据公式，直观的实现是我们在feature的维度进行两次循环，即

```python
v = tf.get_variable('v', shape=[feature_size, filed_size, embedding_size], dtype='float32',
                    initializer=tf.truncated_normal_initializer(mean=0, stddev=0.01))
# shape of [None, 1]
filed_sizeield_aware_interaction_terms = tf.constant(0, dtype='float32')

# build dict to find f, key of feature, value of field
for i in range(feature_size):
    for j in range(i + 1, feature_size):
        filed_sizeield_aware_interaction_terms += tf.multiply(
            tf.reduce_sum(tf.multiply(v[i, feature_size_2_filed_size[i]], v[j, feature_size_2_filed_size[i]])),
            tf.multiply(inputs[:, i], inputs[:, j])
        )
```

因此计算费时。

完整代码参考我的github，
https://github.com/End-the-cold-night/deeprec/blob/master/deeprec/ranking/ctr/model/Ffm.py

调包使用：
 from deeprec.ranking.ctr import Ffm

AFM

03

Attention FM

前面我们介绍分析了FM某个特定的feature，在对其它的feature表示时候采用的是同一个表示，对学习产生了一定的难度。前一个小节介绍了基于FFM的思路，然而，FFM的参数量大，学习速度慢。
另一个思路是我们不在FM的表征上进行区分，而是对交叉的结果进行不同的加权，即AFM。
首先我们直观的看一下AFM的网络结构：

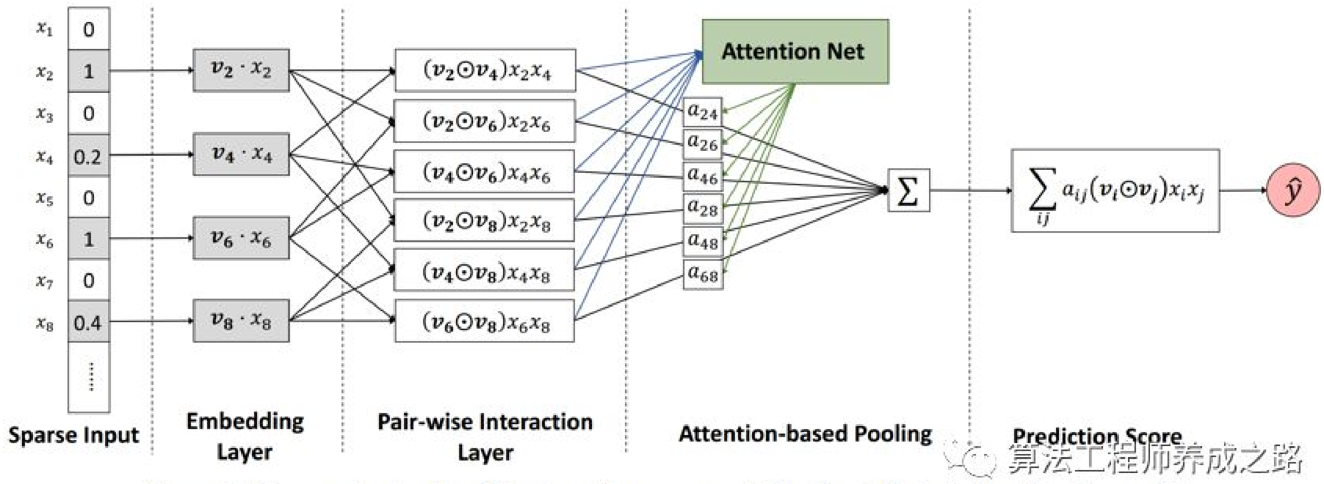

Figure 1: The neural network architecture of our proposed Attentional Factorization Machine model.

不难看出前面embedding层还是取出embedding，然后经过一个pair wise的交叉，然后通过一个网络得到pair wise的权重，即

$$a'_{ij} = \mathbf{h}^T ReLU(\mathbf{W}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j + \mathbf{b}),$$

$$a_{ij} = \frac{\exp(a'_{ij})}{\sum_{(i,j)\in\mathcal{R}_x} \exp(a'_{ij})},$$

之后对结果在feature维度进行求和，即：

$$\hat{y}_{AFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \mathbf{p}^T \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{ij}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j,$$

代码实现：

首先获取pair wise交叉的结果：

```
# element_wise
element_wise_product_list = []
for i in range(field_size):
    for j in range(i + 1, field_size):
        element_wise_product_list.append(tf.multiply(embeddings[:, i, :], embeddings[:, j, :]))  # bs
element_wise_product = tf.stack(element_wise_product_list)  # (fs * fs
element_wise_product = tf.transpose(element_wise_product, perm=[1, 0, 2],name='element_wise_product')
```


定义attention network的权重：

```
weights = dict()
# attention part
glorot = np.sqrt(2.0 / (attention_size + embedding_size))
weights['attention_w'] = tf.Variable(np.random.normal(loc=0, scale=glorot,
                                     size=(embedding_size, attention_size)),
                         dtype=tf.float32, name='attention_w')
weights['attention_b'] = tf.Variable(np.random.normal(loc=0, scale=glorot, size=(attention_size,)),
                         dtype=tf.float32, name='attention_b')
weights['attention_h'] = tf.Variable(np.random.normal(loc=0, scale=1, size=(attention_size,)),
                         dtype=tf.float32, name='attention
weights['attention_p'] = tf.Variable(np.ones((embedding_size, 1)), dtype=np.float32)
```


加权求和：

```
# attention part
num_interactions = int(field_size * (field_size - 1) / 2)
attention_wx_plus_b = tf.reshape(tf.add(tf.matmul(
    tf.reshape(element_wise_product,shape=(-1, embedding_size)),weights['attention_w']),
    weights['attention_b']),shape=[-1, num_interactions, attention_size])
# bs * ( fs * fs * 1 / 2) * as
attention_exp = tf.exp(tf.reduce_sum(tf.multiply(tf.nn.relu(attention_wx_plus_b),
                                                 weights['attention_h']),
                                     axis=2, keep_dims=True))
#  bs * ( fs * fs * 1 / 2) * 1
attention_exp_sum = tf.reduce_sum(attention_exp, axis=1, keep_dims=True)  # bs * 1 * 1
```

完整代码参考我的github，
https://github.com/End-the-cold-night/deeprec/blob/master/deeprec/ranking/ctr/model/Afm.py
调包使用：

```
from deeprec.ranking.ctr import Afm
```

后续会继续介绍其它CTR算法，包括FGCNN，Autoint，DCN等。