计算广告CTR预估系列(十)--AFM模型理论与实践

原创 可爱又迷人的反派角色宁宁 机器学习荐货情报局 2018-06-27

计算广告CTR预估系列(十)--AFM模型理论与实践

一、简介
二、FM
三、AFM
3.1 模型
3.2 模型训练
3.3 过拟合
四、总结
五、代码实践
Reference
计算广告CTR预估系列往期回顾

一、简介

AFM全称是*Attentional Factorization Machine*，和NFM是同一个作者。AFM是在FM上的改进，它最大的特点就是*使用一个attention network来学习不同组合特征的重要性。*

推荐系统或者CTR预估中输入中类别型特征比较多，因为这些类别型特征不是独立的，所以

他们的组合特征就显得非常重要。

一个简单的办法就是给每一个组合特征(cross feature)一个权重。但是这种cross feature-based方法的通病就在于训练集中很多组合特征并没有出现，导致无法有效学习。

FM通过为每一个特征学习一个嵌入向量，也叫做隐向量，通过两个隐向量的内积来表示这个组合特征的权重。但是同样有个问题就是，在预测中有一部分特征是不重要甚至是没用的，它们会引入噪声并对预测造成干扰。对于这样的特征，在预测的时候应该赋予一个比较小的权重，但是FM并没有考虑到这一点。对于不同的特征组合，FM并没有区分它们的权重(可以认为内积之后看成一个组合特征，它们的权重都是1)。

本文通过引入Attention机制，创新新的提出了AFM，用来赋予不同的特征组合不同的重要程度。权重可以在网络中自动学习，不需要引入额外的领域知识。更重要的是，AFM可以

二、FM

FM全称是*Factorization Machine*，形式化公式如下：

$$\hat{y}_{FM}(\mathbf{x}) = \underbrace{w_0 + \sum_{i=1}^{n} w_i x_i}_{\text{linear regression}} + \underbrace{\sum_{i=1}^{n} \sum_{j=i+1}^{n} \hat{w}_{ij} x_i x_j}_{\text{pair-wise feature interactions}} ,$$

Factorization Machine

其中w是两个特征隐向量v的内积。FM有下面两个问题：

1. 一个特征针对其他不同特征都使用同一个隐向量。所以有了FFM用来解决这个问题。
2. 所有组合特征的权重w都有着相同的权重1。AFM就是用来解决这个问题的。

在一次预测中，并不是所有的特征都有用的，但是FM对于所有的组合特征都使用相同的权重。AFM就是从这个角度进行优化的，针对不同的特征组合使用不同的权重。这也使得模型更加*可解释性*，方便后续针对重要的特征组合进行深入研究。

三、AFM

> AFM全称是Attentional Factorization Machine。

3.1 模型

AFM的模型结构如下：

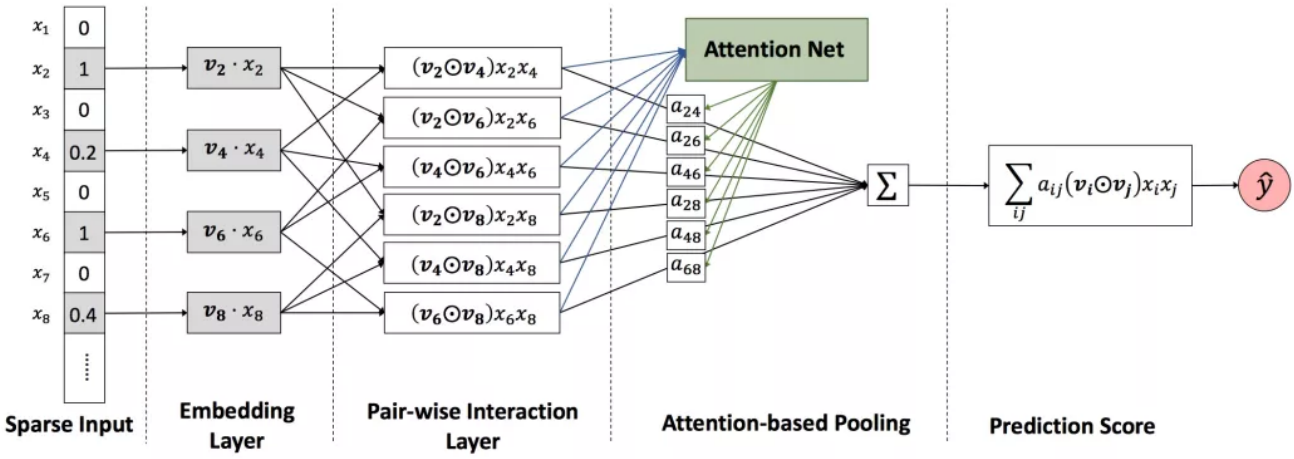

Figure 1: The neural network architecture of our proposed Attentional Factorization Machine model.

AFM模型架构

> 注意，这里面省去了线性部分，只考虑特征组合部分。

*Sparse Input*和*Embedding Layer*和FM中的是相同的，Embedding Layer把输入特征中非零部分特征embed成一个dense vector。下面着重说说剩下的三层。

Pair-wise Interaction Layer:

这一层主要是对组合特征进行建模，原来的m个嵌入向量，通过element-wise product操作得到了 m(m-1)/2 个组合向量，这些向量的维度都是嵌入向量的维度k。形式化如下：

$$f_{PI}(\mathcal{E}) = \{(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j\}_{(i,j)\in\mathcal{R}_x},$$

Pair-wise Interaction Layer

也就是Pair-wise Interaction Layer的输入是所有嵌入向量，输出也是一组向量。输出是任意两个嵌入向量的element-wise product。任意两个嵌入向量都组合得到一个Interacted vector，所以m个嵌入向量得到 m(m-1)/2 个向量。

如果不考虑Attention机制，在Pair-wise Interaction Layer之后直接得到最终输出，我们可以形式化如下：

$$\hat{y} = \mathbf{p}^T \sum_{(i,j)\in\mathcal{R}_x} (\mathbf{v}_i \odot \mathbf{v}_j) x_i x_j + b,$$

Generalize FM

其中p和b分别是权重矩阵和偏置。当p全为1的时候，我们发现这就是FM。这个只是说明AFM的表达能力是在FM之上的，实际的情况中我们还使用了Attention机制。NFM中的Bilinear Interaction Layer也是把任意两个嵌入向量做element-wise product，然后进行sum pooling操作。

Attention-based Pooling Layer:

Attention机制的核心思想在于：当把不同的部分压缩在一起的时候，让不同的部分的贡献程度不一样。**AFM通过在Interacted vector后增加一个weighted sum来实现Attention机制。**形式化如下：

$$f_{Att}(f_{PI}(\mathcal{E})) = \sum_{(i,j)\in\mathcal{R}_x} a_{ij}(\mathbf{v}_i \odot \mathbf{v}_j) x_i x_j,$$

Attention-based Pooling Layer

aij是Attention score，表示不同的组合特征对于最终的预测的贡献程度。可以看到：

1. Attention-based Pooling Layer的输入是Pair-wise Interaction Layer的输出。它包含 m(m-1)/2 个向量，每个向量的维度是k。（k是嵌入向量的维度，m是Embedding Layer中嵌入向量的个数）
2. Attention-based Pooling Layer的输出是一个*k维向量*。它对Interacted vector使用Attention score进行了weighted sum pooling操作。

Attention score的学习是一个问题。一个常规的想法就是随着最小化loss来学习，但是这样做对于训练集中从来没有一起出现过的特征组合的Attention score无法学习。

AFM用一个*Attention Network*来学习。

Attention network实际上是一个**one layer MLP**，激活函数使用*ReLU*，网络大小用*attention factor*表示，就是*神经元的个数*。

Attention network的输入是两个嵌入向量element-wise product之后的结果(interacted vector，用来在嵌入空间中对组合特征进行编码)；它的输出是组合特征对应的Attention score。最后，使用softmax对得到的Attention score进行规范化，*Attention Network*形式化如下：

$$a'_{ij} = \mathbf{h}^T ReLU(\mathbf{W}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j + \mathbf{b}),$$
$$a_{ij} = \frac{\exp(a'_{ij})}{\sum_{(i,j)\in\mathcal{R}_x} \exp(a'_{ij})},$$

Attention Network

总结，AFM模型总形式化如下：

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

AFM模型总形式化

前面一部分是线性部分；后面一部分对每两个嵌入向量进行element-wise product得到Interacted vector；然后使用Attention机制得到每个组合特征的Attention score，并用这个score来进行weighted sum pooling；最后将这个k维的向量通过权重矩阵p得到最终的预测结果。

3.2 模型训练

AFM针对不同的任务有不同的损失函数。

1. 回归问题。square loss。
2. 分类问题。log loss。

论文中针对回归问题来讨论，所以使用的是square loss，形式化如下：

$$L_r = \sum_{x \in \mathcal{T}} (\hat{y}_{AFM}(\mathbf{x}) - y(\mathbf{x}))^2,$$

AFM square loss

模型参数估计使用的是SGD。

3.3 过拟合

防止过拟合常用的方法是Dropout或者L2 L1正则化。AFM的做法是：

1. 在Pair-wise Interaction Layer的输出使用Dropout
2. 在Attention Network中使用L2正则化

Attention Network是一个one layer MLP。不给他使用Dropout是因为，作者发现如果同时在interaction layer和Attention Network中使用Dropout会使得训练不稳定，并且降低性能。

所以，AFM的loss函数更新为：

$$L = \sum_{x \in \mathcal{T}} (\hat{y}_{AFM}(\mathbf{x}) - y(\mathbf{x}))^2 + \lambda ||\mathbf{W}||^2$$

AFM Loss Function

其中W是Attention Network的参数矩阵。

四、总结

AFM是在FM的基础上改进的。相比于其他的DNN模型，比如Wide&Deep，DeepCross都是通过MLP来隐式学习组合特征。这些Deep Methods都缺乏解释性，因为并不知道各个组合特征的情况。相比之下，FM通过两个隐向量内积来学习组合特征，解释性就比较好。

通过直接扩展FM，AFM引入Attention机制来学习不同组合特征的权重，即保证了模型的可解释性又提高了模型性能。但是，DNN的另一个作用是提取高阶组合特征，AFM依旧只

考虑了二阶组合特征，这应该算是AFM的一个缺点吧。

五、代码实践

完成代码、数据以及论文资料请移步github，不要忘记star呦~

> https://github.com/gutouyu/ML_CIA

核心的网络构建部分代码如下：

先准备设置参数，以及初始化Embedding和Linear的权重矩阵：

```
#------hyper parameters------
field_size = params['field_size']
feature_size = params['feature_size']
embedding_size = params['embedding_size']
l2_reg = params['l2_reg']
learning_rate = params['learning_rate']

dropout = params['dropout']
attention_factor = params['attention_factor']

#------build weights------
Global_Bias = tf.get_variable("bias", shape=[1], initializer=tf.constant_initialize
Feat_Wgts = tf.get_variable("linear", shape=[feature_size], initializer=tf.glorot_n
Feat_Emb = tf.get_variable("emb", shape=[feature_size, embedding_size], initializer

#------build feature------
feat_ids = features['feat_ids']
feat_vals = features['feat_vals']
feat_ids = tf.reshape(feat_ids, shape=[-1, field_size])
feat_vals = tf.reshape(feat_vals, shape=[-1, field_size]) # None * F
```

FM的线性部分：

```
# FM部分: sum(wx)
with tf.variable_scope("Linear-part"):
    feat_wgts = tf.nn.embedding_lookup(Feat_Wgts, feat_ids) # None * F * 1
    y_linear = tf.reduce_sum(tf.multiply(feat_wgts, feat_vals), 1)
```

Embedding Layer部分：

```
#Deep部分
with tf.variable_scope("Embedding_Layer"):
```

```
    embeddings = tf.nn.embedding_lookup(Feat_Emb, feat_ids) # None * F * K
    feat_vals = tf.reshape(feat_vals, shape=[-1, field_size, 1]) # None * F * 1
    embeddings = tf.multiply(embeddings, feat_vals) # None * F * K
```

Pair-wise Interaction Layer对每一对嵌入向量都进行element-wise produce:

```
with tf.variable_scope("Pair-wise_Interaction_Layer"):
    num_interactions = field_size * (field_size - 1) / 2
    element_wise_product_list = []
    for i in range(0, field_size):
        for j in range(i + 1, field_size):
            element_wise_product_list.append(tf.multiply(embeddings[:, i, :], embedd
    element_wise_product_list = tf.stack(element_wise_product_list) # (F*(F-1)/2) *
    element_wise_product_list = tf.transpose(element_wise_product_list, perm=[1,0,2]
```

Attention Network用来得到Attention Score:

```
# 得到Attention Score
with tf.variable_scope("Attention_Netowrk"):

    deep_inputs = tf.reshape(element_wise_product_list, shape=[-1, embedding_size])

    deep_inputs = contrib.layers.fully_connected(inputs=deep_inputs, num_outputs=at

    aij = contrib.layers.fully_connected(inputs=deep_inputs, num_outputs=1, activat

    # 得到attention score之后，使用softmax进行规范化
    aij = tf.reshape(aij, shape=[-1, int(num_interactions), 1])
    aij_softmax = tf.nn.softmax(aij, dim=1, name="attention_net_softout") # None *

    if mode == tf.estimator.ModeKeys.TRAIN:
        aij_softmax = tf.nn.dropout(aij_softmax, keep_prob=dropout[0])
```

得到Attention Score之后，和前面的Interacted vector进行weighted sum pooling，也就是Attention-based Pooling Layer:

```
with tf.variable_scope("Attention-based_Pooling_Layer"):
    deep_inputs = tf.multiply(element_wise_product_list, aij_softmax) # None * (F(F-
    deep_inputs = tf.reduce_sum(deep_inputs, axis=1) # None * K Pooling操作

    # Attention-based Pooling Layer的输出也要经过Dropout
    if mode == tf.estimator.ModeKeys.TRAIN:
        deep_inputs = tf.nn.dropout(deep_inputs, keep_prob=dropout[1])

    # 该层的输出是一个K维度的向量
```

Prediction Layer，最后把Attention-based Pooling Layer的输出k维度向量，得到最终预测结果。这一层可以看做直接和一个神经元进行向量。注意这个神经元得到类似logists的值，还不是概率。这个值和后面的FM全局偏置、FM linear part得到最终的logists，然后再通过sigmoid得到最终预测概率：

```
with tf.variable_scope("Prediction_Layer"):
    # 直接跟上输出单元
    deep_inputs = contrib.layers.fully_connected(inputs=deep_inputs, num_outputs=1,
    y_deep = tf.reshape(deep_inputs, shape=[-1]) # None

with tf.variable_scope("AFM_overall"):
    y_bias = Global_Bias * tf.ones_like(y_deep, dtype=tf.float32)
    y = y_bias + y_linear + y_deep
    pred = tf.nn.sigmoid(y)
```

运行结果截图：

```
训练.....
Parsing ./data/tr.mini.libsvm
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:From /home/ubuntu/Example/AFM/AFM.py:103: calling softmax (from tensorflow.python.ops.nn_ops)
Instructions for updating:
dim is deprecated, use axis instead
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into ./model_save/model.ckpt.
INFO:tensorflow:step = 1, loss = 0.7182179
INFO:tensorflow:global_step/sec: 0.430397
INFO:tensorflow:step = 11, loss = 0.63515884 (23.234 sec)
INFO:tensorflow:global_step/sec: 0.821992
INFO:tensorflow:step = 21, loss = 0.60071784 (12.166 sec)
INFO:tensorflow:global_step/sec: 0.840741
INFO:tensorflow:step = 31, loss = 0.54831386 (11.894 sec)
INFO:tensorflow:Saving checkpoints for 40 into ./model_save/model.ckpt.
INFO:tensorflow:Loss for final step: 0.46819106.
```

Train

```
评估.....
Parsing ./data/va.mini.libsvm
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-06-27-10:10:03
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ./model_save/model.ckpt-40
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-06-27-10:10:13
INFO:tensorflow:Saving dict for global step 40: auc = 0.58814, global_step = 40, loss = 0.5559401
INFO:tensorflow:global_step, was: 40
INFO:tensorflow:loss, was: 0.5559400916099548
INFO:tensorflow:auc, was: 0.5881400108337402
```

Evaluate

```
预测.....
Parsing ./data/te.mini.libsvm
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ./model_save/model.ckpt-40
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:0.35452207922935486
```

Predict

Reference

1. Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks
2. https://github.com/lambdaji/tf_repos/blob/master/deep_ctr/Model_pipeline/AFM.py

计算广告CTR预估系列往期回顾

计算广告CTR预估系列(一)--DeepFM理论
计算广告CTR预估系列(二)--DeepFM实践
计算广告CTR预估系列(三)--FFM理论与实践
计算广告CTR预估系列(四)--Wide&Deep理论与实践
计算广告CTR预估系列(五)--阿里Deep Interest Network理论
计算广告CTR预估系列(六)--阿里Mixed Logistic Regression
计算广告CTR预估系列(七)--Facebook经典模型LR+GBDT理论与实践
计算广告CTR预估系列(八)--PNN模型理论与实践
计算广告CTR预估系列(九)--NFM模型理论与实践

机器学习荐货情报局，特别有料！