

初学者系列: FFM: Field-aware Factorization Machines

原创 Sha Li 专知 2019-08-19

导读

点击率 (CTR) 预测在广告行业中起着重要作用。FM、FFM、Deep FM 广泛用于此任务。本文我们主要介绍 FM 的变体, Field-aware Factorization Machines (FFM) 的原理、推导过程, 以及使用 Tensorflow 的简单实现。

获取代码

请关注专知公众号 (点击上方蓝色专知关注)
后台回复“初学者系列FFM”即可获得数据集以及全部代码。



FFM 是 FM 的变体, 在开始介绍 FFM 之前, 我们一起回顾一下 FM 的基本原理, FM 使用分解参数对变量之间的所有交互进行建模, 学习每个特征学习隐层向量, 其输出公式如下:

$$\hat{y} = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^n w_{i,j} x_i x_j$$

在FM中所有的样本都使用同一个V，即对于 $x_{\{11\}}$ 与 $x_{\{12\}}$ 都使用同一个 v_1

FFM与FM的不同 01

FFM提出了Field-aware 的思想，即每一维特征（feature）都归属于一个特定的field，field和feature是一对多的关系。为更好的解释FFM与FM之间的不同，我们以论文中的数据为例：

Clicked	Publisher (P)	Advertiser (A)	Gender (G)
Yes	ESPN	Nike	Male

对于FM每个特征只有一个隐藏向量来学习具有任何其他特征的潜在影响。

$$\phi_{FM}(w, x) = w_{ESPN} \cdot w_{Nike} + w_{ESPN} \cdot w_{Male} + w_{Nike} \cdot w_{Male}$$

w_{ESPN} 用于了解耐克 ($w_{ESPN} \cdot w_{Nike}$) 和男性 ($w_{ESPN} \cdot w_{Male}$) 的潜在影响。然而，由于Nike和Male属于不同的领域，(ESPN, Nike) 和 (ESPN, Male) 的潜在影响可能不同。而对于FFM，每个特征都有几个潜在的向量。

$$\phi_{FFM}(w, x) = w_{ESPN,A} \cdot w_{Nike,P} + w_{ESPN,G} \cdot w_{Male,P} + w_{Nike,G} \cdot w_{Male,A}$$

FFM原理 02

在FFM中，每一维特征 x_i ，针对每一种field(f_j)的特征，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。与FM模型类似，FFM模型方程定义为（为了计算简便只有交叉项）：

$$\hat{y} = \sum_{i=1}^n \sum_{j=i+1}^n \langle v_{i,f_j}, v_{j,f_i} \rangle x_i x_j$$

其中：

- f_i 和 f_j 分别是 x_i 和 x_j 所属的field.

与FM模型不同，在FFM模型方程中，由于隐向量与fields有关故交叉项不可以化简。

在FFM中划分不同的fields是十分重要的一步，对于连续特征与离散特征有不同的field划分方法：

- 类别特征 (Categorical Features) : 采用one-hot编码, 同一种属性的归到一个Field。
- 连续特征 (Numerical Features) : 一个特征对应一个Field。或者对连续特征离散化, 应用类别特征表示方法。

优化 03

我们将FFM问题定义为分类问题, 使用的是logistic loss, 加入正则项后FFM优化目标为:

$$\min \sum_{i=1}^m \log(1 + \exp(-\hat{y}(x^{(i)}), y^i))) + \frac{\lambda}{2} \|v\|_2^2$$

通过梯度下降方法 (SGD) 可以有效地学习FFM的模型参数 $v_{\{i, f_j\}}, v_{\{j, f_i\}}$.

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \hat{y}(x)} * \frac{\partial \hat{y}(x)}{\partial \theta}$$

对于logistic 损失函数

$$\frac{\partial(L)}{\partial \theta} = \frac{e^{-y \cdot \hat{y}}}{1 + e^{-y \cdot \hat{y}}} * y * \frac{\partial \hat{y}(x)}{\partial \theta}$$

为了更好的理解对于不同的field特征是如何计算的, 我们用如下例子进行推倒:

对于如下电影评分数据, 我们以 $x^{(0)}$ 为例:

field	field1(性别)	field2 (电影名称)		
特征		星球大战	星际迷航	泰克尼克号
$x^{(0)}$	男	1	0	0
$x^{(1)}$	女	0	0	1
$x^{(2)}$	男	0	1	0

则 $x^{(0)}$ 的输出 $y^{(0)}$ 为:

$$\hat{y}^{(0)} = v_{1,f2} \cdot v_{2,f1} x_1^{(0)} x_2^{(0)} + v_{1,f2} \cdot v_{3,f1} x_1^{(0)} x_3^{(0)} + v_{1,f2} \cdot v_{4,f1} x_1^{(0)} x_4^{(0)} + \\ v_{2,f2} \cdot v_{3,f2} x_2^{(0)} x_3^{(0)} + v_{2,f2} \cdot v_{4,f2} x_2^{(0)} x_4^{(0)} + v_{3,f2} \cdot v_{4,f2} x_3^{(0)} x_4^{(0)}$$

则对 $v_{1,f2}$ 求偏导得:

$$\frac{\partial \hat{y}^{(0)}}{\partial v_{1,f2}} = v_{2,f1} x_1^{(0)} x_2^{(0)} + v_{3,f1} x_1^{(0)} x_3^{(0)} + v_{4,f1} x_1^{(0)} x_4^{(0)}$$

由于同一个field下只有一个feature的值不是0, 其他feature的值都是0,故:

$$\frac{\partial \hat{y}^{(0)}}{\partial v_{1,f2}} = v_{2,f1} x_1^{(0)} x_2^{(0)}$$

则

$$\frac{\partial \hat{y}}{\partial v_{i,fj}} = v_{j,fi} x_i x_j$$

$$\frac{\partial \hat{y}}{\partial v_{j,fi}} = v_{i,fj} x_i x_j$$

我们选择AdaGrad优化器进行优化, AdaGrad计算第t步之前累加的梯度平方和, 作为学习率的分母, 在训练的每一步自主地选择合适的学习率。具体过程如下:

- 随机采样一个点 (y, x) 来更新参数, 计算一次梯度, 并且只用计算非0值输入对应的参数, 其梯度的计算如下所示

$$g_{i,fj} = \frac{\partial L}{\partial v_{i,fj}} = \frac{e^{-y \cdot \hat{y}}}{1 + e^{-y \cdot \hat{y}}} * y * v_{j,fi} x_i x_j + \lambda v_{i,fj}$$

$$g_{j,fi} = \frac{\partial L}{\partial v_{j,fi}} = \frac{e^{-y \cdot \hat{y}}}{1 + e^{-y \cdot \hat{y}}} * y * v_{i,fj} x_i x_j + \lambda v_{j,fi}$$

- 对每个坐标 $d=1, \dots, k$, 累计梯度的平方和:

$$(G_{i,fj})_d \leftarrow (G_{i,fj})_d + (g_{i,fj})_d^2$$

$$(G_{j,fi})_d \leftarrow (G_{j,fi})_d + (g_{j,fi})_d^2$$

- 最后, 进行梯度更新 (红色框为每一步的学习率)

$$(v_{j,fi})_d \leftarrow (v_{j,fi})_d - \frac{\eta}{\sqrt{(G_{i,fi})_d}} (g_{j,fi})_d$$

实践 04

FFM是一个细化隐向量非常好的方法, 官方也给了实现FFM的libffm。为了更好地理解FFM是如何工作的, 我们用FFM来实现一个简单的二分类。代码主要包含两部分内容, 分别为data.py、FFM.py.

数据处理 (data.py)

原始输入数据以点击率预测为例, 共有16个特征, 三个fields。数据处理主要包括特征提取、field划分以及将数据转化为one-hot类型这三部分内容。

- 类别型特征提取

```
data_path =
'C:/Users/Lenovo/pycharmprojects/PycharmProjects/recommend/FFM/data/dataset.csv'
dataset = pd.read_csv(data_path)#读取训练集数据
dataset['click'] = dataset['click'].map(lambda x: -1 if x == 0 else x)#将训练数据中的点击与否
进行映射, 0变为-1
click = set()
C1 = set()
C16 = set()
C18 = set()
#挑出各个特征的取值集合
data = dataset.copy()#复制训练集

click_v = set(data['click'].values)#以列表返回中所有的值, 删除重复的数据, 变为一个集合{-1,1}
click = click | click_v#并集

C1_v = set(data['C1'].values)
C1 = C1 | C1_v#{1008, 1010, 1001, 1002, 1005, 1007}

C16_v = set(data['C16'].values)
C16 = C16 | C16_v#{480, 50, 250, 36, 20, 90}

C18_v = set(data['C18'].values)
C18 = C18 | C18_v#{0, 1, 2, 3}
```

- field划分

根据每一个特征所属的field, 构造字典。例如在`feature2field={0: 0, 1: 0, 2: 0, 3: 0, ...}`中, value为field, key为变量名的映射; 在`C1.pkl={1008: 0, 1010: 1, 1001: 2, 1002: 3,...}`中, key为变量值, value为变量名的映射。这样就可以通过变量名映射找到各变量值所属的field。

```
#划分fields, fields共有三个, 分别是'C1', 'C18', 'C16'
category_encoding_fields = ['C1', 'C18', 'C16']
feature2field = {}
field_index = 0
ind = 0
for field in category_encoding_fields:
    field_dict = {}
    field_sets = eval(field)#返回表达式的值

    for value in list(field_sets):
        field_dict[value] = ind
        feature2field[ind] = field_index
        ind += 1
    field_index += 1
    with open('C:/Users/Lenovo/pycharmprojects/PycharmProjects/recommend/FFM/data/' +
field + '.pkl', 'wb') as f:
        pickle.dump(field_dict, f)

    with
open('C:/Users/Lenovo/pycharmprojects/PycharmProjects/recommend/FFM/data/feature2field.pk
l', 'wb') as f:
        pickle.dump(feature2field, f)
click_dict = {}
click_sets = click#{1, -1}
for value in list(click_sets):
    click_dict[value] = ind
    ind += 1
with open('C:/Users/Lenovo/pycharmprojects/PycharmProjects/recommend/FFM/data/' + 'click'
+ '.pkl', 'wb') as f:
    pickle.dump(click_dict, f)

# C1.pkl数据为{1008: 0, 1010: 1, 1001: 2, 1002: 3, 1005: 4, 1007: 5}, 1008在
feature2field对应的键为0。。。
# C18.pkl数据为{0: 6, 1: 7, 2: 8, 3: 9}
# C16.pkl数据为{480: 10, 50: 11, 20: 14, 36: 13, 250: 12, 90: 15}
# 以上三个文件为不同特征值对应的索引值
# feature2field{0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 1, 8: 1, 9: 1, 10: 2, 11:
2, 12: 2, 13: 2, 14: 2, 15: 2}
# feature2field不同索引值对应的field
# 对于标签click设置索引值, click.pkl数据为{1: 16, -1: 17}
```

- 数据集构造

在得到特征所属的field后, 还需要对特征值进行one-hot处理才可以构建数据集。


```

#数据处理, 进行特征转换, 并划分各个batch内的数据数量
#将所有的特征转换为one-hot类型
def transfer_data(sample, fields_dict, array_length):
    array = np.zeros([array_length])
    for field in fields_dict:
        # get index of array
        if field == 'click':
            field_value = sample[field]
            ind = fields_dict[field][field_value] #得到索引
            if ind == (array_length - 1):
                array[ind] = -1
            else:
                array[ind + 1] = 1
        else:
            field_value = sample[field]
            ind = fields_dict[field][field_value]
            array[ind] = 1
    return array

#划分各个batch内的数据数量, 还是一次传入64个数据
def get_batch(x, batch_size, index):
    start = index * batch_size
    end = (index + 1) * batch_size
    end = end if end < x.shape[0] else x.shape[0]
    return x.iloc[start:end, : ] #选取x中的start到end行

```

经过处理后的输入x如下图所示, 同一个field的特征值只有一个为0

```

[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

```

构造模型 (FFM.py)

- 权重变量初始化 (b,w1,v)

```

k = 6 #隐向量个数
f = 3 #field的个数
p = 16#特征数
learning_rate = 0.1
batch_size = 64
l2_reg_rate = 0.001
feature2field = None
checkpoint_dir='C:/Users/Lenovo/pycharmprojects/PycharmProjects/recommend/FFM/saver'
training = True
epoch = 1

#定义权重以及偏置变量，并进行初始化

def createTwoDimensionWeight(input_x_size,field_size,vector_dimension):
    v= tf.get_variable('v', shape=[p, f, k], dtype='float32',
                       initializer=tf.truncated_normal_initializer(mean=0, stddev=0.01)) #
    shape=[p,f,k]
    return v
def createOneDimensionWeight(input_x_size):
    w1=tf.get_variable('w1', shape=[p, 1],
                       initializer=tf.truncated_normal_initializer(mean=0, stddev=0.01))
    return w1
def createZeroDimensionWeight():
    b = tf.get_variable('bias', shape=[1],
                        initializer=tf.zeros_initializer()) # b形状为1维，初始化为0
    return b

```

- 定义模型输出

#定义模型，计算模型输出

```
def inference(X, feature2field, b, w1, v):

    with tf.variable_scope('linear_layer'):

        # shape of [None, 1]
        linear_terms = tf.add(tf.matmul(X, w1), b) # 线性部分w1*x+b
        print('self.linear_terms:')
        print(linear_terms)
    # 定义交叉项参数v
    with tf.variable_scope('interaction_layer'):
        field_cross_interaction = tf.constant(0, dtype='float32')
        # 每个特征
        # feature2field={0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 1, 8: 1, 9: 1, 10: 2, 11:
        2, 12: 2, 13: 2, 14: 2, 15: 2}
        for i in range(p):
            for j in range(i + 1, p):
                # print('i:%s,j:%s' % (i, j))
                vifj = v[i, feature2field[j]] # 找到xj对应的field
                vjfi = v[j, feature2field[i]]
                vivj = tf.reduce_sum(tf.multiply(vifj, vjfi)) # 两个矩阵中各自元素相乘
                xixj = tf.multiply(X[:, i], X[:, j])
                field_cross_interaction += tf.multiply(vivj, xixj)
        field_cross_interaction = tf.reshape(field_cross_interaction, (batch_size, 1)) #
        转化为64行1列的数组
        print('self.field_cross_interaction:')
        print(field_cross_interaction)
    y_out = tf.add(linear_terms, field_cross_interaction) # 输出
    return y_out
```

- 加载数据

在进行训练之前需要先加载训练数据集，并将数据通过调用get_batch () 函数，获得一个batch 内传入的数据，再将每个batch内的输入特征通过transfer_data () 转化为one-hot类型。

```
#加载数据，划分了训练集以及测试集
def loaddata(data_path):
    dataset = pd.read_csv(data_path)
    dataset['click'] = dataset['click'].map(lambda x: -1 if x == 0 else x)
    traindata, testdata = train_test_split(dataset, test_size=0.4, random_state=0)
    # 加载feature2field，用来对应不同特征的field
    with open(

'C:/Users/Lenovo/pycharmprojects/PycharmProjects/recommend/FFM/data/feature2field.pkl',
'rb') as f:
        feature2field = pickle.load(f)
    # 加载各个field的特征值
    fields = ['C1', 'C18', 'C16', 'click']
    fields_dict = {}
    for field in fields:
        with open(
            'C:/Users/Lenovo/pycharmprojects/PycharmProjects/recommend/FFM/data/' +
field + '.pkl','rb') as f:
            fields_dict[field] = pickle.load(f)
#fields_dict={'C18': {0: 6, 1: 7, 2: 8, 3: 9}, 'click': {1: 16, -1: 17}, 'C16': {480: 10,
50: 11, 36: 13, 20: 14, 250: 12, 90: 15}, 'C1': {1008: 0, 1010: 1, 1001: 2, 1002: 3,
1005: 4, 1007: 5}}

    return traindata, testdata, feature2field,fields_dict
```

```

#构造数据集，对各个特征进行one-hot转换
def dataset(data, fields_dict):
    all_len = max(fields_dict['click'].values()) + 1 # 18
    cnt = data.shape[0] // batch_size
    for i in range(epoch):
        # 在一个epoch内遍历所有的数据
        for j in range(cnt):
            # 数据转换，将数据转换为one-hot类型
            dataset = get_batch(data, batch_size, j) # 一次传入traindata的64行，依次向下
            actual_batch_size = len(dataset) # 还是64
            batch_x = []
            batch_y = []
            for k in range(actual_batch_size):
                sample = dataset.iloc[k, :] # click      -1
                                     # C1      1005
                                     # C18      0
                                     # C16      50

                array = transfer_data(sample, fields_dict, all_len) # 每一行的特征都转换为
one-hot

                batch_x.append(array[:-2]) # 前面16个是特征
                # 最后一位即为label, [-1]:label=0; [1]:label=1
                batch_y.append(array[-1]) # 最后一个标签

            batch_x = np.array(batch_x)
            batch_y = np.array(batch_y)

            y_int = batch_y.reshape(len(batch_y), 1) # 将label转化为64行1列

    return batch_x, y_int, cnt

```

tips: get_batch () 与transfer_data () 在data.py中已经定义过。

- 优化

在该部分主要进行梯度计算与损失函数计算（加入了正则项）

```

if __name__ == '__main__':
    #获取数据
    data_path =
    'C:/Users/Lenovo/pycharmprojects/PycharmProjects/recommend/FFM/data/dataset.csv'
    traindata, testdata, feature2field, fields_dict = loaddata(data_path)
    train_x, train_y, cnt = dataset(traindata, fields_dict)
    test_x, test_y, test_cnt = dataset(testdata, fields_dict)
    #定义占位符
    x = tf.placeholder(tf.float32, [batch_size, p])
    y = tf.placeholder(tf.float32, [None, 1])
    #调用函数获得变量
    b = createZeroDimensionWeight()
    w1 = createOneDimensionWeight(p)
    v = createTwoDimensionWeight(p, f, k)
    #定义损失函数以及优化器
    lambda_w = tf.constant(0.001, name='lambda_w')
    lambda_v = tf.constant(0.001, name='lambda_v')
    y_out = inference(x, feature2field, b, w1, v)
    loss = tf.reduce_mean(tf.log(1 + tf.exp(-y * y_out))) # 损失函数
    l2_norm = tf.reduce_sum(
        #正则项
        tf.add(
            tf.multiply(lambda_w, tf.pow(w1, 2)),
            tf.reduce_sum(tf.multiply(lambda_v, tf.pow(v, 2)), axis=[1, 2])
        )
    )
    loss = loss+l2_norm

    global_step = tf.Variable(0, trainable=False)
    # 计算梯度
    opt = tf.train.GradientDescentOptimizer(learning_rate) # 随机梯度下降算法
    trainable_params = tf.trainable_variables()
    print(trainable_params) # 查看b, w1, v
    gradients = tf.gradients(loss, trainable_params) # 求梯度,即文中的 $g_{\{i, f_j\}}$ ;  $g_{\{j, f_i\}}$ 
    clip_gradients, _ = tf.clip_by_global_norm(gradients, 5)
    train_op = opt.apply_gradients(zip(clip_gradients, trainable_params),
    global_step=global_step)
    #设置GPU
    gpu_config = tf.ConfigProto()
    gpu_config.gpu_options.allow_growth = True # 当使用GPU时候, Tensorflow运行自动慢慢达到最大
    GPU的内存

```

- 开始训练

```

with tf.Session(config=gpu_config) as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(tf.local_variables_initializer())
    #训练
    if training:
        for i in range(epoch):
            #在一个epoch内遍历所有的数据
            for j in range(cnt):
                cost, _, step = sess.run([loss, train_op, global_step], feed_dict={
                    X: train_x,
                    y: train_y
                })
                if j % 100 == 0:
                    print('After {0} training steps , and the loss is %s' % (j,
                        cost))
                    save(sess, checkpoint_dir)

```

训练结果:

```

After {0} training steps , and the loss is 0.6879269
After {100} training steps , and the loss is 0.434821
After {200} training steps , and the loss is 0.49895376
After {300} training steps , and the loss is 0.40871555
After {400} training steps , and the loss is 0.39381155
After {500} training steps , and the loss is 0.39977896
After {600} training steps , and the loss is 0.49957126
After {700} training steps , and the loss is 0.3967054
After {800} training steps , and the loss is 0.42074686
After {900} training steps , and the loss is 0.30288962
After {1000} training steps , and the loss is 0.51065135
After {1100} training steps , and the loss is 0.4343705
After {1200} training steps , and the loss is 0.357248
After {1300} training steps , and the loss is 0.4500237
After {1400} training steps , and the loss is 0.44454095
After {1500} training steps , and the loss is 0.37505034
After {1600} training steps , and the loss is 0.37782806
After {1700} training steps , and the loss is 0.4543309
After {1800} training steps , and the loss is 0.44171908

```

- 测试

在测试部分, 输出了测试集的损失函数以及预测值。

```

else:
    restore(sess, checkpoint_dir)
    result, cost = sess.run([y_out, loss], feed_dict={
        X: test_x, y: test_y})

    print(result)

```

测试结果如下:


```

[ -1.297571 ]
[ -1.6127795 ]
[ -1.6127795 ]
[ -0.594898 ]
[ -1.297571 ]
[ -1.297571 ]
[ -1.6127795 ]]
0.5157192

```

参考链接:

https://github.com/sladesha/machine_learning/tree/master/FFM

-END-

专 · 知

专知，专业可信的人工智能知识分发，让认知协作更快更好！欢迎登录www.zhuanzhi.ai，注册登录专知，获取更多AI知识资料！



欢迎微信扫一扫加入**专知人工智能知识星球群**，获取**最新AI专业干货知识教程视频资料**和与**专家交流咨询**！



请加**专知小助手微信**（扫一扫如下二维码添加），加入**专知人工智能主题群**，咨询**技术商务合作**~