项目代码说明

模型总体说明

融合DeepFM与Item2Vec的方法。将'user_id'分别与'sku_id','vender_id','brand','shop_id','cate'五种类别特征交互，对于5种类别特征的其中一个，都用Word2vec训练出的64维Embedding向量表示，将结果加入DeepFM的DNN中，其余部分保持DeepFM原有模型不变。

程序入口，主函数:`main.py`

模型数据输入格式

- [ ] **Xi**: *[[ind1_1, ind1_2, ...], [ind2_1, ind2_2, ...], ..., [indi_1, indi_2, ..., indi_j, ...], ...]*
    - *indi_j* is the feature index of feature field *j* of sample *i* in the dataset
- [ ] **Xv**: *[[val1_1, val1_2, ...], [val2_1, val2_2, ...], ..., [vali_1, vali_2, ..., vali_j, ...], ...]*
    - *vali_j* is the feature value of feature field *j* of sample *i* in the dataset
    - *vali_j* can be either binary (1/0, for binary/categorical features) or float
- [ ] **Xo**:[[sku_emb_1_1, sku_emb_2_1, ..sku_emb_64_1,vender_emb_1_1...vender_emb_64_1,....],

  [sku_emb_1_2, sku_emb_2_2, ..sku_emb_64_2,vender_emb_1_2...vender_emb_64_2,....],

  [...]

  ...]
    - 每一行为一个样本的所有Item2Vec特征，特征维数为 64*5=320
- [ ] **y**: target of each sample in the dataset (1/0 for classification, numeric number for regression)

以上都是由数组形式通过`feed_dict`载入到模型之中进行训练。

网络静态图构建

```
class DeepFM(object):
    # 输入参数和初始化定义在说明中省略
    def _build_graph(self):
        self.add_input()
        self.inference()

    def add_input(self):
        self.index = tf.placeholder(tf.int32, shape=[None, self.field_num],
name='feat_index')  # (batch, F)
        self.x = tf.placeholder(tf.float32, shape=[None, self.field_num],
name='feat_value')  # (batch, F)
        # Item2Vec Emb
        self.I2v_Emb = tf.placeholder(tf.float32, shape=[None, self.I2v_num],
name='I2v_Emb')  # (batch, I)
```

```
        self.y = tf.placeholder(tf.float32, shape=[None], name='input_y')
        self.is_train = tf.placeholder(tf.bool)

    def inference(self):
        with tf.variable_scope('first_order_part'):
            first_ord_w = tf.get_variable(name='first_ord_w', shape=
[self.feat_num, 1], dtype=tf.float32)
            first_order = tf.nn.embedding_lookup(first_ord_w, self.index)  #
(batch, F, 1)
            first_order = tf.reduce_sum(tf.multiply(first_order,
tf.expand_dims(self.x, axis=2)), axis=2)  # (batch, F)
        with tf.variable_scope('emb_part'):
            embed_matrix = tf.get_variable(name='second_ord_v', shape=
[self.feat_num, self.vec_dim], dtype=tf.float32)
            embed_v = tf.nn.embedding_lookup(embed_matrix, self.index)  #
(batch, F, K)
            embed_x = tf.multiply(tf.expand_dims(self.x, axis=2), embed_v)  #
(batch, F, K)
        with tf.variable_scope('second_order_part'):
            sum_emb_square = tf.square(tf.reduce_sum(embed_x, axis=1))  #
(batch, K)
            square_emb_sum = tf.reduce_sum(tf.square(embed_x), axis=1)  #
(batch, K)
            second_order = 0.5 * (sum_emb_square - square_emb_sum)
            fm = tf.concat([first_order, second_order], axis=1)  # (batch,
F+K)
        with tf.variable_scope('dnn_part'):
            embed_x = tf.layers.dropout(embed_x, rate=self.dropout_rate,
training=self.is_train)  # (batch, F, K)
            in_num_1 = self.field_num * self.vec_dim
            dnn_1 = tf.reshape(embed_x, shape=(-1, in_num_1))  # (batch,
in_num1)
            dnn  = tf.concat([dnn_1,self.I2v_Emb],axis=1) # (batch, in_num) =
(batch,F*K+I)
            in_num = in_num_1 + self.I2v_num
            for i in range(len(self.dnn_layers)):
                out_num = self.dnn_layers[i]
                w = tf.get_variable(name='w_%d' % i, shape=[in_num, out_num],
dtype=tf.float32)
                b = tf.get_variable(name='b_%d' % i, shape=[out_num],
dtype=tf.float32)
                dnn = tf.matmul(dnn, w) + b
                dnn = tf.layers.dropout(tf.nn.relu(dnn),
rate=self.dropout_rate, training=self.is_train)
                in_num = out_num
        with tf.variable_scope('output_part'):
            in_num += self.field_num + self.vec_dim # in_num =
F+K+self.dnn_layers[-1]
            output = tf.concat([fm, dnn], axis=1)
```

```
            proj_w = tf.get_variable(name='proj_w', shape=[in_num, 1],
dtype=tf.float32)
            proj_b = tf.get_variable(name='proj_b', shape=[1],
dtype=tf.float32)
            self.y_logits = tf.matmul(output, proj_w) + proj_b
        self.y_hat = tf.nn.sigmoid(self.y_logits)
        self.pred_label = tf.cast(self.y_hat > 0.5, tf.int32)
```

数据加载

`utils/ReadData.py`

- 比赛数据集加载、预处理、构建训练样本、人工特征工程: `class jdata_process():`
- 将数据调整为如上面第二项中所示的**输入格式**:`class DataParser(object):`。在这个步骤中调用`class FeatureDictionary(object):`是用来对每个样本在每个特征的field的维数中的第几位进行编码。

Item2Vec训练

`utils/Item2vec.py`

以用户为中心，对于'sku_id','vender_id','brand','shop_id','cate'之中的每一项，将用户的历史序列['sku_id1','sku_id2'...'sku_idm']看成一个句子。（这里我选用的数据是行为在2018-03-15日之后的）

- 用`model = Word2Vec(sentence, size=L, window=10, min_count=1, workers=10, iter=10)`进行训练。
- 将解决存储为`.pkl`文件格式，以便模型加载数据时读取。

  `out_df.to_pickle("../temp/" + f + "_I2v_" + str(L) + ".pkl")`

  训练结果以文件形式保存在`temp/`位置中

Reference

[1]https://github.com/ChenglongChen/tensorflow-DeepFM

[2]https://github.com/charleshm/deep-ctr