知乎 首发于 机器学习深度学习应用

推荐算法注意点和DeepFM工程化实现

奔奔

机器学习、深度学习、推荐算法、反作弊、nlp

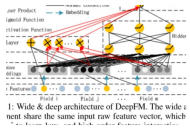
关注他

题图和文章无关，添加一个美女是为了骗点击。图片找了很久，才找到一个没有版权问题的并且自己看着顺眼的，希望大家喜欢！如果大家从文章分享的干货中有所收获，请点个赞！

缘起

今年疫情期间开始优化公司的推荐系统，因为DeepFM具有使用线性特征、低阶交叉特征和高阶特征的优点，决定采用此算法试试能否提高线下的auc和线上的CTR预估。DeepFM算法介绍见：

奔奔：深入浅出DeepFM
zhuanlan.zhihu.com



低一些。主要存在两个问题，在实际过程中，无法保存auc最优的模型，early stopping也不能保证停在效果最好的阶段；在线上预测阶段是不能按照文件的方式去读取。主要是针对以上两个问题，进行改造，实现了工程化上线。效果方面：点击率PV提升了2.67%；点击率UV提升了3.64%；平均点击数提升了4.53%。推荐系统实际工程中需要注意样本、特征、算法等方方面面的问题，下面开始介绍整个项目。

项目背景

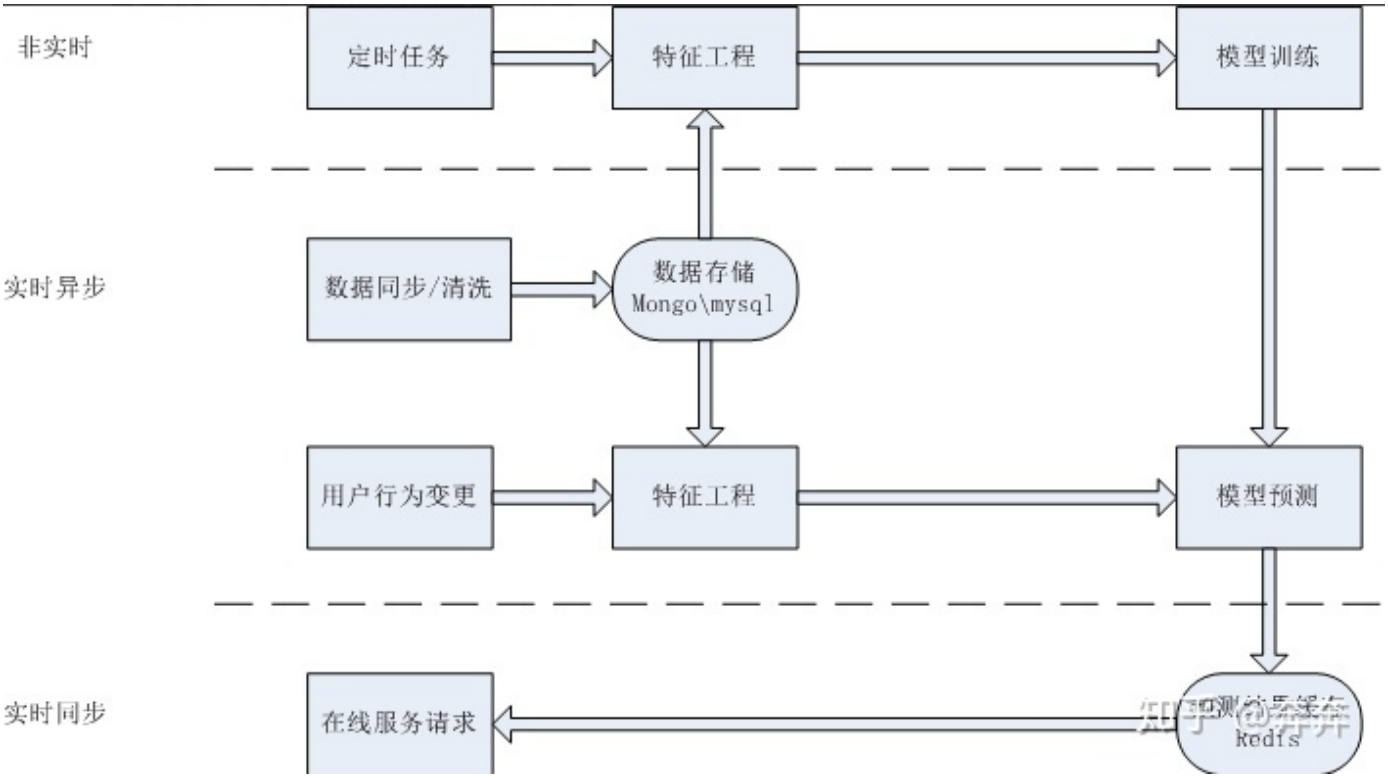
混沌大学APP（以下简称APP）是一个提供哲科思维和创新商业的课程在线学习软件，在线视频学习是APP提供的最重要的业务功能。APP内提供上百门十几分钟至几小时不等的长视频课程，为了用户更快的发现合适自己的课程，以及拥有更好的学习体验，APP提供了课程推荐的功能。

算法页面

由于app推荐的物品通常是十几分钟到几小时的长视频课程学习，且课程数量相对较少，学员存在断续观看、重复观看的需求；并且几百门课如果采用feed流方式（推荐视频不重复）很容易就把所有课程内容刷没了。因此没有使用feed流的推荐模式，而是使用了topN推荐。

算法架构

由于推荐的物品课程总量仅为百级，因此没有做召回而实现了排序+重排序。为了解决有限的计算资源与实效性之间的矛盾，采用了实时异步的架构，即在用户有行为时去为用户预测一个新的课程列表并缓存，在用户请求时，才将这个缓存列表发给用户，即拉模式（pull），整体架构如下：



工程架构

下面从样本、特征工程、算法改造、重排序、探索位介绍整个项目流程。

样本

模型学习的内容是样本，模型效果的上限在样本，所以样本的设计和选择非常重要。如何选择正负样本？样本冲突了怎么办？如何使得样本表达的内容丰富，受到噪声干扰比较小？在线系统的特征获取比离线训练某些特征数据有延迟怎么办？

面对以上问题，整个样本的生成采取宁缺毋滥的原则，对于不确定是否正确的数据，均不采纳。例

正样本的选择相对比较容易，点击即为正。负样本的选择相对复杂一点：课程曝光，并不代表用户注意到了。因此选择用户在推荐列表最下面的一个点击位置以上的曝光作为负样本区域。例如以下展示列表和点击动作情况（最下一个点击位为7），使用3、7为正样本，1、2、4、5、6为负样本。而8、9、10等位置虽然曝光但用户可能并未看到，丢弃该数据。正负样本的比例一般设置为1:1或者2:3。

1	未点击
2	未点击
3	点击
4	未点击
5	未点击
6	未点击
7	点击
8	未点击
9	未点击
10	未点击

一条请求数据返回的样本列表点击情况

样本冲突

课程可以重复推荐，这样就存在用户选择前后矛盾的情况，即对于同一个课程上次用户选择点击，而这次选择不点击，或者反过来上次选择不点击，这次选择点击。

对于一个时效性要求较高的系统，将这两种情况的数据都作为样本加入系统，可以增加模型对时效性特征的理解。由于用户可能某个时间段未关注、或者被其他课程干扰、或者某个时刻的心理状态等等问题存在。在宁缺毋滥的原则下，选择了一刀切去掉负样本的方式，即24小时之内，存在正负样本冲突的，仅保留正样本。同时如果24小时之内有多个负样本，仅保留最后也就是最新的负样本。

样本下采样

由于用户不同天以及天内对推荐系统的使用频率不同，会导致整个样本数据就会倾向于样本时间段内使用频率加高的那些样本用户，这对其他的用户是不公平的。同理，不同的课程的每天曝光量也有很大差别。因此分别以用户和课程为单位对整体样本进行下采样，使得样本数量的比例更加平滑一些。

数据穿透问题

袭，而在上线预测时，标准答案还存在于尚未发生的未来的，模型此时得不到标准答案，预测结果就很差。

由于CTR预估对实时性要求高，实际过程中存在另外一种数据穿透的情况，线上数据延迟带来的穿透。即，在离线训练阶段，如果选取样本时刻之前的所有特征，这些特征本来是没有穿透的，但是上线阶段，由于前端、后台、数据系统等等存在一些延迟，最近时间的一些特征在预测时并没有流入推荐系统，导致线上预测阶段，模型拿到的是残缺的数据。这样会导致模型离线阶段效果还不错，线上阶段预测效果就不好的情况。

面对这个问题，在算法离线训练阶段，就不考虑样本最近一段时间的特征数据，即让模型只使用残缺的数据学习，逼迫模型从残缺的数据中发掘数据关联，即不依赖线上容易发生延迟的数据部分。这个处理会使得线下AUC指标降低，即降低了算法上限，但在数据延迟的情况下有更好的健壮性。

特征工程

特征延用旧算法中的大部分，只有word2vec类型的特征没有加入。本身这些文本类别的特征都能在embedding层被编码表示，也加入课程属于的类别、用户侧点击、搜索、分享等特征做成长兴趣和短兴趣两种。对于数据归一化，分桶等常规操作就不介绍了。

由于DeepFM输入数据需要有索引和值的需求，所以将数据分为4类。一类是连续的单值：课程的播放比率，下载次数，距今时间等；一类是连续的一列多值：用户侧老师的刻画，课程标签等；一类是离散单值：是否有研习社权限，课程是否免费；一类是离散的一列多值：用户和课程的联系，课程的多个老师等。

DeepFM

在实际使用的DeepFM算法的过程中，有些特征会比较稀疏，并且存在一列多值的情况和权重共享的要求，需要解决这三个方面的实际问题。

1. 稀疏要求：课程、用户标签、课程标签、老师、关键词都很多并且稀疏，可以采用编码方式，自动转成embedding。
2. 一列多值：比如说用户点击过的课程是一个序列；用户可以有多个标签；用户可以和课程建立多个联系等，将同一列多个元素的embeding做累加。
3. 权重共享：用户对课程点击或者观看时长都是对课程的刻画；用户画像和课程画像共用同一个词表。

正好看到石塔西的代码，能够解决以上的问题，但是还存在两个问题需要解决：

知乎 首发于 机器学习深度学习应用

因此需要对模型训练部分和模型预测部分的数据输入进行修改和优化,将TensorFlow高阶的API改成低阶的API，实现更加灵活的控制。

原文采用tf_utils.to_sparse_input_and_drop_ignore_values(dense_Xs[k])将稠密文件矩阵转成稀疏矩阵。按照输出的格式采用sparse_element（下面代码中有）方法和tf.compat.v1.SparseTensorValue构建起一行一行的读入方式便于线上实时预测使用：

```
class FeaturePredictV2(object):

    @classmethod
    def sparse_element(cls,suffix, flag, max_tokens):
        #构造稀疏特征，按照行处理成需要的格式
        values_lst = []
        indices_lst = []
        row_length = len(suffix)
        for i in range(row_length):
            for j in range(len(suffix[i])):
                values_lst.append(suffix[i][j])
                indices_lst.append([i, j])
        indices = np.array(indices_lst, dtype=np.int)
        values = np.array(values_lst, dtype=np.int32) if flag else np.array(values_lst
        dense_shape = np.array([row_length, max_tokens])
        return [indices, values, dense_shape]

    @classmethod
    def get_processed_data(cls,data_pre):
        #数据读入并处理
        X = {}
        name_list = [t[0] for t in COLUMNS_MAX_TOKENS]
        df = pd.DataFrame(data_pre, columns=name_list)
        for colname, max_tokens in COLUMNS_MAX_TOKENS:
            kvpairs = [i.split(",")[:max_tokens] for i in df[colname]]
            ids = []
            vals = []
            for lines in kvpairs:
                id = []
                val = []
                try:
                    for line in lines:
```

```
                    print('get_processed_data error {}'.format(sys.exc_info()))
                    print(err)
                    traceback.print_exc()
                ids.append(id)
                vals.append(val)
            ids_elements = cls.sparse_element(ids, 1, max_tokens)
            vals_elements = cls.sparse_element(vals, 0, max_tokens)
            X[colname + "_ids"] = tf.compat.v1.SparseTensorValue(ids_elements[0], ids_
            X[colname + "_values"] = tf.compat.v1.SparseTensorValue(vals_elements[0],

        return X


if __name__ == '__main__':

    data_pre = []
    with open('data/train/train_st.dat') as f:
        for line in f.readlines()[:3]:
            line_list = line.split(" ")[1:]
            processed_line = [i.rstrip("\n") for i in line_list]
            # print(processed_line)
            data_pre.append(processed_line)

    print(FeaturePredictV2.get_processed_data(data_pre))
```

模型训练部分保存auc最高的部分：主要是构图和保存模型两段代码。

构图部分如下，将原代码中高阶API的核心部分用低阶API重新写成，将sess、init、features、labels、train_graph、cost、predictions、embedding_table灵活提取出来，可以供整个过程的使用。

```
    def build_model(self, params):
        """
        构图
        :param name:
        :param params:
        :return:
        """
        train_graph = tf.Graph()
```

```
        features[c + "_values"] = tf.compat.v1.sparse_placeholder(tf.float32,

    labels = tf.compat.v1.placeholder(tf.int32, shape=[None, 1])
    x = tf.compat.v1.placeholder(tf.int32)
    y = tf.compat.v1.placeholder(tf.int32)

    for featname, featvalues in features.items():
        if not isinstance(featvalues, tf.SparseTensor):
            raise TypeError("feature[{}] isn't SparseTensor".format(featname))

    embedding_table = self.build_embedding_table(params)

    linear_logits = self.output_logits_from_linear(features, embedding_table,

    bi_interact_logits, fields_embeddings = self.output_logits_from_bi_interac


    dnn_logits = tf.cond(tf.less(x, y), lambda: self.f1(fields_embeddings, par
                         lambda: self.f2(fields_embeddings, params))

    general_bias = tf.compat.v1.get_variable(name='general_bias', shape=[1], i
    # logits = linear_logits + bi_interact_logits
    logits = linear_logits + bi_interact_logits + dnn_logits
    logits = tf.nn.bias_add(logits, general_bias)  # bias_add，获取broadcasting
    # del features
    logits = tf.reshape(logits, shape=[-1])
    predictions = tf.sigmoid(logits)
    labels = tf.cast(labels, tf.float32)
    labels = tf.reshape(labels, shape=[-1])
    cost = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(logits=logits, labels=labels))
    optimizer = params['optimizer'].minimize(cost, global_step=tf.compat.v1.tr
    # auc =tf.compat.v1.metrics.auc(labels, predictions)
    sess = tf.compat.v1.Session(graph=train_graph)
    init = tf.group(tf.compat.v1.global_variables_initializer(), tf.compat.v1.
    return Struct(sess=sess, init=init, features=features, labels=labels, trai
                  predictions=predictions, x=x, y=y,
                  optimizer=optimizer, embedding_table=embedding_table)
```

训练保存auc最优模型部分：

写成想要的，如果auc高于之前最高的就当前的覆盖之前最好的模型，只保留一个最优的模型。

```
def train(self, train_data, test_data, save_path=model_dir):
      train_len_data = len(train_data)
      train_total_batch = train_len_data // self.batch_size
      print("train_data length:{},toal_batch of training data:{}".format(train_len_d
      test_len_data = len(test_data)
      test_total_batch = test_len_data // self.batch_size
      print("test_data length:{},toal_batch of testing data:{}".format(test_len_data
      self.model = self.build_model(self.param)
      self.model.sess.run(self.model.init)
      max_score = 0.0
      fl_train = Feature_load(self.train_eval, self.num_epochs, self.batch_size, sel
      train_input = fl_train.input_fn(train_data)
      fl_eval = Feature_load(self.train_eval, self.num_epochs, self.batch_size, self
      test_input = fl_eval.input_fn(test_data)
      for epoch in range(self.num_epochs):
          loss_static = []
          all_predictions=[]
          all_labels=[]
          for i in range(train_total_batch):
              train_feed_dict = self.get_input_data(train_input, True)
              # loss, _ = self.model.sess.run([self.model.cost, self.model.optimizer
              loss, _,predictions,labels = self.model.sess.run([self.model.cost, sel
                                                  feed_dict=train_feed_dict)
              # print(auc)
              loss_static.append(loss)
              all_predictions.extend(predictions)
              all_labels.extend(labels)
          avg_loss = float(sum(loss_static)) / len(loss_static)
          r = EvaMgr(['AUC'])
          test_auc_entity=r.evaluate(all_labels,all_predictions)
          test_auc =test_auc_entity['AUC']
          print("Epoch:{}/{}".format(epoch, self.num_epochs),
                "Train loss:{:.3f} Train AUC:{:.3f}".format(avg_loss,test_auc))

          eva, avg_test_loss = self.evaluate(test_total_batch, test_input, ['AUC', '
          score = eva['AUC']
          print("Epoch:{}/{} Test loss:{:.3f} Test AUC: {:.3f}".format(epoch, self.n
          print(eva)
```

```
if not os.path.exists(save_path):
    os.mkdir(save_path)
self.save(saved_model)
```

以上的三段代码还需要其他才能跑起来，读者可以自行构造一些数据并结合石塔西公开的代码，由于公司限制不能公开全部代码，但是把主要的思路和改造部分贴了出来，供大家参考。

重排序

推荐系统线上预测通常对于相似的课程会得到比较相似的得分，在推荐列表中展示时会表现出相邻的课程通常会比较相似。因此。对CTR预估的推荐列表进行了重排序，如果相邻的两个课程有相同的教师或者学籍，则对排名考虑的课程与后续课程互换位置进行微调，使得展示列表更加平衡。经过测试，调整后的列表指标上与调整前没有明显差异，但是展示效果更好。

探索位

推荐系统与热单都会倾向不再推质量较低的，大部分用户不太喜欢或者曝光少的的课程。长期会导致一些课程得不到曝光的机会，甚至彻底沉没。

为了解决该问题，增加了一个探索推荐位，强制抽选一个该用户曝光最少的课程（曝光比例越低，被选中的概率越大），放在前十位推荐的课程中。该方法会在一定程度上降低点击率，但是会增加课程的整体曝光度，降低马太效应，提升课程多样性和新颖性。

线上下线一致性验证

CTR预估的bug相对于业务系统其实比较难以发现和测试，因为CTR预估的bug通常仅导致线上指标的下降而非功能的缺失和错误。

算法模型通常分为离线训练阶段和上线预测阶段，离线阶段需要批量对数据进行特征处理，而预测阶段需要对数据单个特征处理，基于处理速度和计算性能的考虑，以及两个阶段对于特征数据时间的处理不同，通常会使用两套不同的代码。离线代码可以通过离线指标保证其正确性，而在线代码如果有bug很难通过在线指标进行验证，这是由于：

1. 在线指标同时受很多其他因素影响。
2. 在线指标需要积累一定的数据才能统计，此时才发现问题，实际上有bug的代码已经运行了一段

1. 对于特征处理离线部分和在线部分尽可能多的使用相同的子函数（虽然整体流程不一致，但是部分子过程是相同的），对于不一致的部分例如样本时间的处理，可以作为子函数参数传递。
2. 在上线之前，分别使用离线部分和在线部分的代码对相同的样本进行预测，如果样本的浮点数得分完全相同（小数点后若干位），则可以认为在线部分没有bug。

评价指标

评价指标对推荐系统有重要意义，是衡量推荐算法优劣的重要依据，根据业务场景需要采用了四个评价指标来衡量CTR预估，并以前三个为主，第四个做参考。

1、平均点击率PV = 总点击PV/总展示PV
2、平均点击率UV = 总点击UV/总展示UV
3、平均点击数 = 总点击PV/总展示UV
4、平均观看时长 = 总播放时长/播放UV

效果

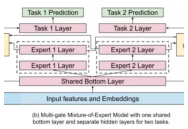
指标增量		
点击率PV	点击率UV	平均点击数
2.67%	3.64%	4.53%

点击率增量指标

深入一步

在点击率提升后就做推荐系统的多目标优化，现在做点击和时长的优化，具体的思想工具参见下面的文章，全面总结了多目标的各种情况。由于点击和时长相关关系还是很明显，所以采用阿里的**ESMM。**

奔奔：推荐系统中的多目标学习
zhuanlan.zhihu.com



参考文献