1. 图灵联邦

1.1. 简介

比赛任务是：在特定的时间和地点下，预测用户对某个视频的点击率CTR。https://www.turingtopia.com/competitionnew/disDetail?discussId=2cc516a3187b426fbb1a795ce6f4619a&cid=e4880352b6ef4f9f8f28e8f98498dbc4

官方提供给我们的数据包括，用户的属性信息，比如：用户的设备ID，设备品牌，设备的操作系统，网络类型等；视频的信息主要包括视频ID；此外，还有一些上下文信息，包括用户所在的经纬度信息（地点）、视频推荐位置pos和视频曝光给用户的时间戳 ts，用户点击视频的时间戳 timestamp 。 （17个字段信息）

过程：

①首先，我们参考了Youtube DNN做特征的思路，为了得到每个视频的Embedding向量，选取每个用户最近点击的前N个（20）视频生成一条用户的点击序列，通过Word2vec生成视频的Embedding（8维）向量。在这个过程中，因为Word2vec模型过分依赖于视频的出现顺序，而在短时间内视频的点击顺序与用户画像的刻画不是强关联的，所以我们借鉴Deepwalk的思想，将含有N个视频的点击序列随机打乱几次，这样每个用户就会生成多条点击序列数据，然后再输入到Word2vec中，得到每个视频的Embedding。

② 然后计算用户的Embedding向量，这里我们结合了阿里的DIN的思想，引入注意力机制（Attention），将用户点击视频的Embedding向量进行加权求和，得到用户的Embedding表示。我们将点击的视频与待预测视频的Embedding做内积得到权重。

③此外，我们也根据业务场景提取了一些统计特征，比如针对设备ID（用户），统计用户前一天的点击次数，浏览量，点击率等特征；针对视频ID，统计其前一小时的曝光量、被点击次数等，因为这些特征都是连续型的统计信息，我们先使用Lightgbm训练这些特征，得到了连续型特征的重要性，对这些特征的重要性进行排序，然后选取了前top-k个重要连续特征进行保留。对这些特征使用GBDT做离散化（Facebook），输入到DeepFM进行训练。DeepFM模型训练的同时，我们将之前Word2vec提取的用

户Embedding与DeepFM的Embedding层拼接，得到最终的预测结果。

④最终线上的成绩，AUC 0.815。（最好0.83左右）（将400多个特征降到了200多个特征。）

在用Word2Vec思想构建用户的embedding的时候，**有一个问题是，用户观看视频的顺序并没有很强烈的关系，而Word2Vec则比较重视视频点击的顺序（邻近的词之间关联很强），导致最后用户的embedding会有一些偏差。**为此，我们采用随机游走的思想，对于一个用户构建多个随机的视频点击序列，交给模型去学习，使得最后得到的embedding能够对用户画像描绘的更准确。

使用GBDT的好处：利用GBDT可以自动进行特征筛选和特征组合，进而生成新的离散特征向量。因为回归树中每个节点的分裂是一个自然的特征选择的过程，而多层节点的结构则对特征进行了有效地自动组合。所以可以非常高效地解决棘手的特征选择和特征组合的问题。

实验中设置30棵树，深度为8。每颗树都相当于一个类别特征，每棵树的叶子结点数相当于特征值的个数。

1.2. 总体细节

1. 数据规模：训练集数据1000多万条，测试集数据360w条，用户数量11万，视频数量10万。（正负样本比例不均衡，1:9）
2. 线上成绩：AUC 0.815
3. 提升最大的是Embedding特征，能够提2，3个百分点。其他重要的特征包括：视频的曝光时间差ts，视频曝光的位置pos相关特征。
4. 难点：视频的刻画，因为官方数据中，视频相关的字段只有视频ID（newsid），其余诸如视频类型、时长、内容等属性信息都没有，因此很难对视频进行刻画，所以我们想到**通过用户的点击序列来得到视频的Embedding**。
5. 数据集划分：选取8、9号两天的数据作为训练集数据，选取10号数据做验证集数据，然后保留模型的最优迭代次数，利用模型对线上数据进行预测。

1.3. 数据预处理

1. 通过对数据进行查看和分析，**发现代表用户的字段有两个，分别是设备id和用户的注册id**，通过对比可以发现数据中的每一个设备id和每一个用户的注册id几乎是一一对应的（因为注册id是有Nan空值的），**因此这两个字段略显重复，所以只保留一个设备id即可**。

2. **设备品牌**['device_vendor']中，**既有大写又有小写**（Xiaomi和xiaomi），因此可以将它们统一转换成小写，以便后续对此特征进行label_encoder编码。（该特征作用不大）

```
str = "www.runoob.com"
print(str.upper())          # 把所有字符中的小写字母转换成大写字母
print(str.lower())          # 把所有字符中的大写字母转换成小写字母
print(str.capitalize())     # 把第一个字母转化为大写字母，其余小写
print(str.title())          # 把每个单词的第一个字母转化为大写，其余小写
```

3. 为了更具体的表征用户的位置，**对原始数据提供的经纬度信息进行处理**，包括以下两个步骤：

（1） 首先对经度取前7位（总共9位），维度取前6位（总共8位），这样可以避免用户位置细微变动带来的噪声影响，相当于做了一个简单地平滑；

（2） 另外我们将经纬度字符串进行拼接来表示用户的具体位置；

```
df['lng_lat'] = df['lng'].astype('str') + '_' + df['lat'].astype('str') # 经纬度连起来代表确切地理位置，（没进行平滑处理），然后编码即可
```

4. **对视频曝光时间进行展开，得到时间戳的年月日时分秒信息，以便后续特征工程使用。**

```
train_df['date'] = pd.to_datetime(
    train_df['ts'].apply(lambda x: time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(x / 1000))) # 对视频曝光时间进行展开
)
df['hour'] = df['date'].dt.hour
df['total_hour'] = df['hour'] + 24 * (df['day'] - 8) # 距离八号0点，过了多少小时，即曝光时间距离8号的时长
```

5. **对类别特征重新编码。**

```
cate_cols = [
    'deviceid', 'newsid', 'pos', 'app_version', 'device_vendor',
    'netmodel', 'osversion', 'device_version', 'lng_lat'
]
for f in cate_cols:
    map_dict = dict(zip(df[f].unique(), range(df[f].nunique()))) # 对类别特征重新编码，unique：特征值，nunique:特征值的个数
    df[f] = df[f].map(map_dict).fillna(-1).astype('int32') # 进行编码并将缺失值填充为 -1
```

1.4. 特征工程

1. **历史信息特征组**

- 针对设备id（deviceid），相当于对每个用户，统计他前一天的点击次数、浏览量、点击率。**（尝试更细粒度的统计，比如前一小时、前六小时等，但是更细粒度的统计在这该数据上效果不佳，因为该数据集在时间上的分布并不均匀，粒度太细的话，会造成大量样本的特征缺失，反而影响最终的效果。）**

```
# 对前一天的点击次数进行统计，对click_df操作
tmp = click_df.groupby([f, 'day'], as_index=False)['id'].agg({f + '_prev_day_click_count': 'count'}) # 统计用户在某一天的点击次数
tmp['day'] += 1
df = df.merge(tmp, on=[f, 'day'], how='left')
df[f + '_prev_day_click_count'] = df[f + '_prev_day_click_count'].fillna(0) # 用户前一天没有点击，用0填充

# 对前一小时的点击次数进行统计（也可以试试统计后一小时的点击次数，但涉及到label的统计量+时间穿越带来的问题是标签泄露）
```

```
# 一小时也可以拓展到两小时、6小时等； 统计用户在前一小时的点击量
tmp = click_df.groupby([f, 'total_hour'], as_index=False)['id'].agg({'{}_prev_total_hour_click_count'.format(f): 'count'})
tmp['total_hour'] += 1 # +2就是对前两个小时统计， -2的话就是对后两小时统计，比如得到第8小时的点击量，8-2=6，即把第八小时的点击量merge到第6小时上
df = df.merge(tmp, on=[f, 'total_hour'], how='left')
df['{}_prev_total_hour_click_count'.format(f)] = df['{}_prev_total_hour_click_count'.format(f)].fillna(0) # 前一小时没点击，0填充

# 对前一天的曝光量进行统计，对df操作
tmp = df.groupby([f, 'day'], as_index=False)['id'].agg({f + '_prev_day_count': 'count'})
tmp['day'] += 1
df = df.merge(tmp, on=[f, 'day'], how='left')
df[f + '_prev_day_count'] = df[f + '_prev_day_count'].fillna(0)

# 计算前一天的点击率，用户i前一天的点击率 = 用户i前一天的点击量/（用户i前一天的曝光量 + 所有用户前一天曝光量的均值）
df[f + '_prev_day_ctr'] = df[f + '_prev_day_click_count'] / (df[f + '_prev_day_count'] + df[f + '_prev_day_count'].mean())

# 对前一小时的曝光量进行统计
tmp = df.groupby([f, 'total_hour'], as_index=False)['id'].agg({f + '_total_hour_count': 'count'})
tmp['total_hour'] += 1
df = df.merge(tmp, on=[f, 'total_hour'], how='left')
df.rename(columns={f + '_total_hour_count': '{}_prev_total_hour_count'.format(f)}, inplace=True)
df['{}_prev_total_hour_count'.format(f)] = df['{}_prev_total_hour_count'.format(f)].fillna(0)

# 计算前一小时的点击率，用户i前一小时的点击率 = 用户i前一小时的点击量/（用户i前一小时的曝光量 + 所有用户前一小时曝光量的均值）
df['{}_prev_total_hour_ctr'.format(f)] = df['{}_prev_total_hour_click_count'.format(f)] / (
  df['{}_prev_total_hour_count'.format(f)] + df['{}_prev_total_hour_count'.format(f)].mean())

# 对后一小时的曝光量进行统计（因为不涉及label的统计量穿越也无妨，但实际应用场景中该类特征无法获取）
```

```
tmp['total_hour'] -= 2 # 后两个小时，例如得到8点的曝光量，但是-2后代表6点的曝光量，所以代表的是对6点两个小时后的曝光量进行统计。
df = df.merge(tmp, on=[f, 'total_hour'], how='left')
df.rename(columns={f + '_total_hour_count': '{}_next_total_hour_count'.format(f)}, inplace=True) # 没有的位置Nan
```

- **针对视频id（newsid），相当于对每个视频，统计它前一小时的曝光量、被点击次数。**

历史特征主要用过去一个时间单位的数据进行统计，然后作为当前时刻的特征。由于数据中前一天老用户占比较多，所以本方案大量构造了前一天数据统计特征。在这部分特征中涉及到了各种点击率的构造，而点击率（=点击量/展现量）计算会用到标签数据，**因此相比于全局构造点击率，构造前一天的点击率避免了标签泄露和数据穿越的问题。**

数据穿越：对后一小时的曝光量进行统计（因为不涉及label的统计量穿越也无妨，但实际应用场景中该类特征无法获取）

标签泄露： 在统计全局点击率时需要使用未来的标签，所以标签泄露。

- deviceid：对设备id统计它前一天的点击次数、浏览量、点击率
- deviceid，hour：对设备id统计它前一天每个小时（0-23）的点击率
- deviceid，netmodel：对设备id统计它前一天不同网络环境下的点击率
- newsid：对视频id统计它前一小时的被点击次数，点击率

2. **构建count特征**，即以不同的时间单位对训练集三天的数据进行一个全局统计，时间单位包括：十分钟、小时、天、全局，例如：对三天的数据集统计每个视频曝光量总量，统计这三天用户更倾向于在哪个时间，哪个网络环境下使用app。所以：count特征主要反应用户偏好属性，具有一定曝光度的作用（今天哪个视频的曝光量大），但是这部分count特征往往会出现数据穿越问题。（用了未来的信息，来对当天的样本进行预测）

```
df[f + '_count'] = df[f].map(df[f].value_counts()) # 新增count特征,即类别中特征值出现的次数统计，如不同视频id出现的总次数
```

3. **对反应时间（点击时间 - 曝光时间）特征进行统计**

用户点击视频的反应时间：用户点击视频的时间戳timestamp和视频被曝光给用户的时间戳ts的差值。二者的差值可以表示**用户点击视频的反应时间**，如果反应时间越短就说明用户越喜欢该类型的视频。针对设备id和视频id来构造统计特征，构造方式：max/min/mean/std。比如统计当前设备点击视频的平均反应时间，最大反应和最小反应时间。

```
# f = 'deviceid';  click_df:用户的点击df，只有点击的数据。
click_df['exposure_click_gap'] = click_df['timestamp'] - click_df['ts'] # 从曝光到点击的时间差（反应时间）
tmp = click_df.groupby([f, 'day'], as_index=False)['exposure_click_gap'].agg({
        f + '_prev_day_exposure_click_gap_max': 'max', f + '_prev_day_exposure_click_gap_min': 'min',
        f + '_prev_day_exposure_click_gap_mean': 'mean', f + '_prev_day_exposure_click_gap_std': 'std'})
# 将day加1再merge即表示前一天，同理减1再merge表示后一天，但会穿越，而且测试集没有后一天
tmp['day'] += 1 # 本来第8天的数据统计，加一就变成第9天的数据，代表前一天的反应时间的数据。 疑问：第八天的前一天数据怎么得到?
# 而且对于没有点击的用户就不存在反应时间， 统计信息特征的位置全为 空值Nan
df = df.merge(tmp, on=[f, 'day'], how='left')
```

4. **视频曝光时间差统计：用户当前曝光时间与前一次曝光时间差、与前N次曝光时间的时间差，与下一次曝光时间的时间差、与下N次曝光时间差**

- 正常历史统计：用户前一次点击视频与当前点击视频的时间间隔、前一次浏览与当前浏览的时间间隔等。（本方案对devceid、newsid、pos等特征和相互之间的组合进行了前1次、前2次、前3次、前5次、前10次的曝光时间差统计）

```
tmp = sort_df.groupby(f) # 对用户进行groupby
# 上一次曝光到当前的时间差 = 当前曝光时间 - 上一次曝光时间
sort_df['{}_prev_exposure_ts_gap'.format(f)] = tmp['ts'].shift(0) - tmp['ts'].shift(1) # shift(N)
```

- 时间穿越统计：本次数据集存在一个巨大的leak（泄露），那就是对曝光时间差进行穿越统计，时间穿越其实就是指在特征构造的时候用到了未来的信息。

  正常历史统计是统计前n次与当前的时间差，而穿越统计则统计后n次与当前的时间差（**用户后一次点击视频与当前点击视频的时间间隔**）。加上这组穿越统计特征之后，分数可以暴涨十几个百分点，然而这都是没有意义的，**因为在真实业务场景下是不可能获取到未来的信息的**。

  - **曝光时间戳之间的gap（后一个ts减去当前ts）很能反映用户是否点击观看了，这是一个穿越特征。**

  （基本构造方法就是计算距离下一次视频曝光的时间差。这么做的原因也很好理解，假如一个人点击了某个视频，那么必然会观看一段时间，那么距离下一次视频的曝光就会久一点，曝光时间ts 差值也较大。相反，连续两次视频的曝光时间间隔应该很小。）

```
# 当前到下一次曝光的时间差 = 下一次的曝光时间 - 当前曝光时间
sort_df['{}_next_exposure_ts_gap'.format(f)] =
tmp['ts'].shift(-1) - tmp['ts'].shift(0)

tmp2 = sort_df[
   [f, 'ts', '{}_prev_exposure_ts_gap'.format(f),
'{}_next_exposure_ts_gap'.format(f)] # 取四列特征
    ].drop_duplicates([f, 'ts']).reset_index(drop=True) # 去除重复数据
df = df.merge(tmp2, on=[f, 'ts'], how='left') # 用户id，  当前曝光时间 ts，拼接上次曝光时间差和下次曝光时间差
```

5. **交叉特征组**

- 我们从离散特征中挑选出了一些较为重要的特征进行两两交叉，进行特征组合，比如：设备id、视频id、推荐位置pos、网络环境。
- 本方案从中挑选了5个较为重要的特征进行两两交叉，这5个特征分别是deviceid、newsid、pos、netmodel、lng_lat。
- 比如设备id和视频id交叉，可以统计各个设备与各个视频的共现次数、为每个设备推荐不同视频的数量、每个视频在其中的次数占比等。

- ```
  # 共现次数，例如视频j曝光给用户i的次数。f = ‘deviceid', col = 'newsid'
   df = df.merge(df.groupby([f, col], as_index=False)['id'].agg({
        '{}_{}_count'.format(f, col): 'count' # 共现次数，例如视频j曝光给用户i的次数。
      }), on=[f, col], how='left')

  # 比例偏好：视频j曝光给用户i的次数 / 用户i获得的总曝光次数 = 占的比例
  df['{}_{}_count_ratio'.format(col, f)] = df['{}_{}_count'.format(f, col)] / df[f + '_count'] # 用户i总共有多少条数据就代表获得的总曝光次数
  ```

5. **Embedding特征**：先得到用户的点击序列列表，将里面的点击序列打乱，每个用户有多个点击序列，将这些序列输入到Word2vec中。model中包含每个视频的embedding，将用户点击列表中的视频的embedding求平均得到用户的embedding，但实际上使用attention，注意力得分是由当前待预测物品与点序列中物品的内积得到，所以用户的embedding实际上是变化的（随着待预测物品而变化）。

   从gensim库中调用word2vec模型，将所有点击序列列表sentence输入模型， 设置embedding大小，窗口大小window，视频的出现次数不得小于5次，确定使用skip-gram还是cbow，使用分层softmax还是负采样。

   ```
   from gensim.models import Word2Vec
   def emb(df, f1, f2):
       emb_size = 8
       print('========================================= {} {} ========================================='.format(f1, f2))
       tmp = df.groupby(f1, as_index=False)[f2].agg({'{}_{}_list'.format(f1, f2): list}) # 得到每个用户的曝光列表
       sentences = tmp['{}_{}_list'.format(f1, f2)].values.tolist() # 得到列表，列表中包含[曝光列表]
       del tmp['{}_{}_list'.format(f1, f2)]
       for i in range(len(sentences)): # 用户数量
   ```
   ```

```
        sentences[i] = [str(x) for x in sentences[i]]

    # sentences是列表中套列表，size词向量的维度，window窗长，在一个sentence中当前词和预测词的最大距离；
    # min_count最小频次/数，忽略掉出现次数小于该值的所有词；sg 训练的算法，1是skip-gram; otherwise CBOW.
    # hs 如果是1，训练模型中应用hierarchical softmax，如果是0，且`negative`非0，那么将采用负采样
    # negative整数，如果大于0，负采样，意思是引入多少个噪声词，通常是5~20个；是0的话，将不用负采样。hs=0, negative=5,
    # seed : int,用于每个词的初始化向量随机数种子
    model = Word2Vec(sentences, size=emb_size, window=5, min_count=5, sg=0, hs=1, seed=2019)
    emb_matrix = [] # 指的是用户的embedding矩阵，每一行代表用户的embedding
    for seq in sentences:
        vec = []
        for w in seq:# 视频在用户的曝光序列中
            if w in model: # 视频w在model中
                vec.append(model[w]) # 向vec中添加视频w的embedding
        if len(vec) > 0: # 对视频的embedding求平均得到用户的embedding
            emb_matrix.append(np.mean(vec, axis=0))
        else: # 如果没有则 用户embedding为0
            emb_matrix.append([0] * emb_size)
    for i in range(emb_size): # 8
        tmp['{}_{}_emb_{}'.format(f1, f2, i)] = emb_matrix[:, i] # 得到不同1-8embedding大小的
    del model, emb_matrix, sentences
    tmp = reduce_mem(tmp)
    print('runtime:', time.time() - t)
    return tmp


emb_cols = [
    ['deviceid', 'newsid']
#     ['deviceid', 'lng_lat'],
#     ['newsid', 'lng_lat'],
    # ...
]
```

```
for f1, f2 in emb_cols:
    df = df.merge(emb(sort_df, f1, f2), on=f1, how='left') # f1 = 'deviceid', f2 = 'newsid'
    df = df.merge(emb(sort_df, f2, f1), on=f2, how='left')
```

6. **效果不明显的特征**

- 比如数据集给出了用户的设备信息：设备品牌、设备操作系统，对这些特征进行处理，交互等操作，可能由于其自身的重要性就不大，所以加入这些处理后的特征后，提升效果不明显。
- 将数据集的经纬度信息转化成具体的城市，也没有效果。