特征工程｜特征获取、特征规范和特征存储

原创 Thinkgamer 搜索与推荐Wiki 2020-07-27

收录于话题

#Thinkgamer 11 #特征工程 92 #精品小系列内容 25

> 基础数据是特征的基石，因此在考虑特征构建的时候一定要注意基础数据的完备性，同时获取特征之后特征的命名一定要规范，特征的存储设计也一定要符合自己的业务需求。

特征获取

特征往往是依赖于基础数据而存在的，这里的基础数据包括用户注册时填写的内容信息，平台中内容上传时填写的属性信息，用户在平台内产生的各种行为信息。而后期我们所有构建的特征都是依赖于这三大类数据，用户维度特征依赖用户内容信息，事物维度特征依赖事物属性信息，用户和事物之间的交互特征依赖于用户的各种行为，像一些经过更高层次的加工信息（比如embedding特征、id类特征、时间序列特征等）其实底层都离不开我们所定义的三大维度。

特征的获取其实在很大程度是依赖底层数据的，这就要求平台内部要建立完备的数据获取体系和存储体系，方便后期进行数据的加工。基础数据从实时性上是可以分为离线数据和实时数据的，离线数据通常指的是当天之前产生的数据行为集合，而实时数据通常指的是当天内产生的数据行为集合。当然离线数据和实时数据可以通过不同的时间窗口进行更细粒度的划分（如果不作特别说明，下文涉及到的离线数据指当天以前的数据集合，实时数据指当天内的数据集合）。

用户在APP内产生的数据信息第一时间是会同步到相应的业务库中（一般是Mysql），然后我们的数据处理团队会将这些数据中的离线信息同步到数据平台上（一般是分布式存储平台上或者数据仓库中），当然这里在同步的过程中是会对数据做基本的处理和加工，行成格式化的数据，这一整串的流程则会被固化成定时的调度任务执行。对于用户当天产生的实时数据，在记录业务库的同时，也会上报一份到消息通道中（比如Kafka），然后通过实时的数据处理技术将数据写到对应的存储空间中，这里要注意的是，实时数据一般是有业务需求的情况下才会触发实时数据的处理，避免造成资源的浪费。比如业务侧需要实时获取用户的浏览事物ID序列数据，这时候我们可以通过以下流程进行相应的数据处理：

- 平台将用户浏览事物数据进行上报
- 实时处理技术对该份数据进行解析和格式化，得到用户ID，事物ID序列数据
- 将该份数据实时更新到储存库中
- 业务侧从该存储库中取出该份数据

离线数据存储到分布式存储平台之后，进行相应的特征加工和处理，转变成我们需要的特征数据格式。实时数据则通过实时数据处理技术直接加工成我们需要的特征数据格式。

特征规范

在特征的处理过程中，要保证的是数据的可读性高，可用性性强，这里边包括特征的命名规范和特征域聚合。

比如对于用户维度的属性信息特征，通常的做法是将其聚合到用户维度，这样方便我们使用和加载（我们总不能把用户维度的特征放到事物维度里），而对于特征的命名也要注意，通常使用的是特征对应的英文名，比如用户性别，在聚合到用户维度之后可以用gender表示，用户的年龄则可以用age表示。

对于一些复杂的特征，比如用户7天内浏览某事物的次数，可以用 user_see_item_count_7，用户3天内下单的次数，可以用user_order_item_count_3。在进行特征命名的过程中要避免怕名字长而简化命名，这样使用方则很难直观的去理解特征，下面给出一些通用的命名规则：

- 不要使用拼音，尽量使用特征含义对应的英文名表示。就像用户的性别你用xingbie来表示一样，总感觉怪怪的。
- 命名过程中尽量不要出现特殊的字符。一般情况下会使用下划线或者驼峰进行多个单词的拼接，比如上边说的用户7天内浏览某事物的次数，你用user@see@item@count@7表示是不是也挺另类的。
- 特征命名格式一致性。比如上边提到的user_see_item_count_7和user_order_item_count_3，保证了格式的一致性，如果使用user_see_item_count_7和userOrderItemCount3则会给人不严谨的感觉。

- 在保证含义明确的前提下简化命名。比如user_see_item_count_7，也可以用user_see_item_count_in_seven_days，后者相对就臃肿许多。

大致规则如上，当然每个公司每个团队都会有自己的命名规范和习惯，我们在平时的工作中要见微知著，学以致用。

特征存储

特征在产出之后是供我们使用的，在推荐场景中，特征的使用主要分为两个地方：

- 线下模型训练使用
- 线上模型预测使用

对于模型训练而言，不讲究特征的实时性，通常特征会存储在分布式存储平台上（比如hdfs上），当我们训练模型时可以直接读取hdfs上的数据。对于模型预测而言，需要实时加载特征，这时候如果还是存储在hdfs上的话就不能满足实时性了，这种情况下一般将特征存储在分布式的数据库中（比如redis、mongo、mysql），具体使用哪种数据库，要综合考虑公司的技术栈、是否能满足业务需求和数据库本身的稳定性。

在具体的特征存储过程中，通常要考虑读和写的并发操作造成数据库的IO开销增大的情况，这时候要考虑对特征进行聚合写入（比如用户维度的特征放在一个集合下）、建立不同的特征数据库集群，让不同类型不同业务的特征存储在不同的地方等方式（比如离线特征放一个集群，因为是常读少写型，对于数据库的性能要求就低一些，而对于实时类特征，因为是常读常写型，则要求数据库的性能高一些）。

总之我们要本着减少IO网络开销、节省资源的目标去设计存储结构，从而最大限度的发挥价值。

The end