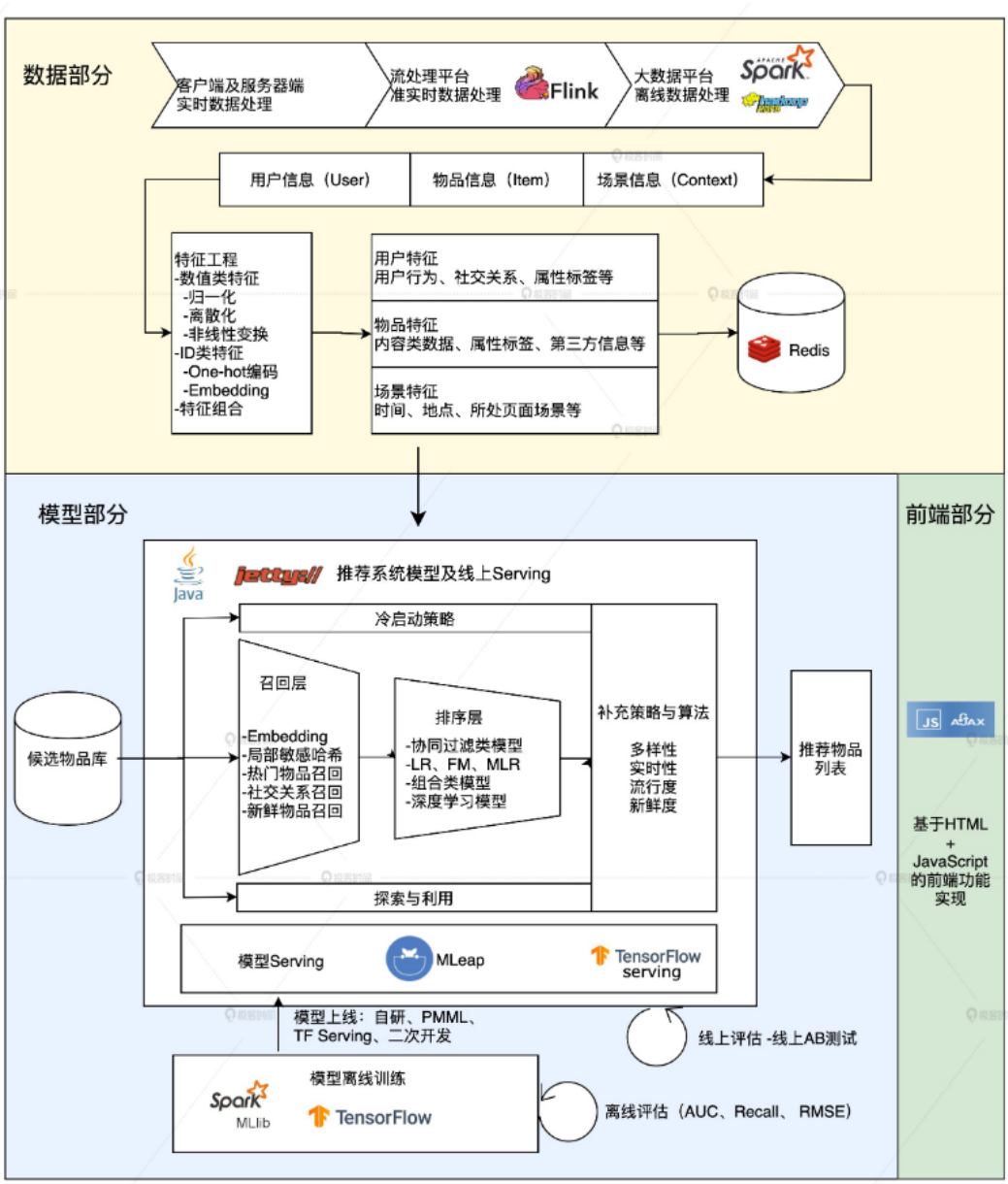
深度学习推荐系统

基础架构篇

SparrowRecsys的架构



特征工程篇

特征工程：有哪些有用特征可以挑选?

- 原则：具体信息转换成抽象特征的过程会造成信息损失，需要保留推荐环境中的有用信息
- 推荐系统中的常用特征
  - 用户行为数据：显性反馈行为与隐性反馈行为

业务场景	显性反馈行为	隐性反馈行为
电子商务网站	对商品的评分	点击、加入购物车、购买等
视频网站	对视频的评分、点赞等	点击、播放、播放时长等
新闻类网站	赞、踩等行为	点击、评论等
音乐网站	对歌曲、歌手、专辑的评分	点击、播放、收藏等

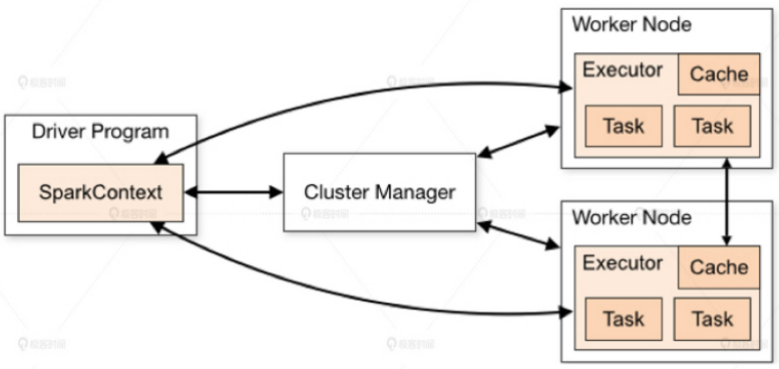
- 用户关系数据：强关系（“关注”，“好友关系”）和弱关系（“互相点赞”，“同看一部电影”）
- 属性、标签类数据：Multi-hot，知识图谱.....

主　体	类　别	来　源
用户	人口属性数据（性别、年龄、住址等）	用户注册信息、第三方DMP（Data Management Platform、数据管理平台）
	用户兴趣标签	用户选择
物品	物品标签	用户或者系统管理员添加
	物品属性（例如，商品的类别、价格；电影的分类、年代、演员、导演等信息）	后台录入、第三方数据库

- 内容类数据：大段的描述型文字、图片，甚至视频；需要通过自然语言处理、计算机视觉等技术手段提取关键内容特征，再输入推荐系统
- 场景信息（context）：最常用的上下文信息是“时间”和通过 GPS、IP 地址获得的“地点”信息

特征处理

- Spark
  - 架构：Spark 程序由 Manager node（管理节点）进行调度组织，由 Worker Node（工作节点）进行具体的计算任务执行，最终将结果返回给 Drive Program（驱动程序）。在物理的 worker node 上，数据还会分为不同的 partition（数据分片），partition 是 Spark 的基础数据单元



  - 最关键的过程：理解哪些是可以纯并行处理的部分，哪些是必须 shuffle（混洗）和 reduce 的部分
  - 计算过程总结：Stage 内部数据高效并行计算，Stage 边界处进行消耗资源的 shuffle 操作或者最终的 reduce 操作
- One-hot编码处理类别型特征
  - 特征处理的目的：把所有的特征全部转换成一个数值型的特征向量
  - One-hot：把所有其他维度置为 0，单独将当前类别或者 ID 对应的维度置为 1 的方式生成特征向量
  - SparrowRecsys：使用 Spark 的机器学习库 MLlib 来完成 One-hot 特征的处理
  - Multi-hot 编码：multiHotEncoderExample的实现
- 数值型特征的处理 - 归一化和分桶
  - 特征的尺度：我们希望把两个特征的尺度拉平到一个区域内，通常是[0,1]范围，这就是所谓**归一化**
  - 特征的分布：特征值分布可能极不均匀，可以使用**分桶**来解决

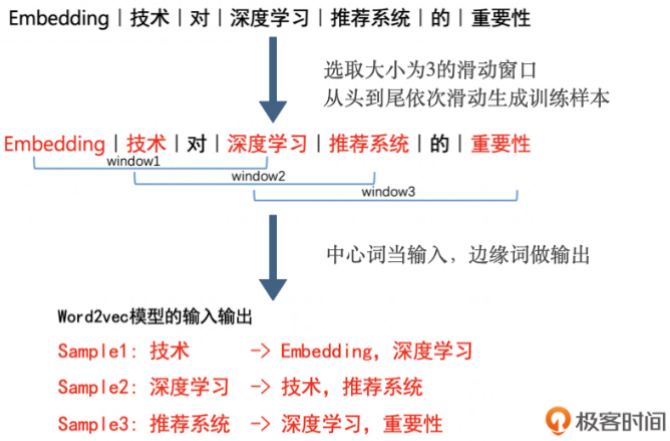
- 分桶（Bucketing）：将样本按照某特征的值从高到低排序，然后按照桶的数量找到分位数，将样本分到各自的桶中，再用桶 ID 作为特征值
- Spark MLlib：使用MinMaxScaler 和 QuantileDiscretizer来进行归一化和分桶的特征处理

Embedding入门

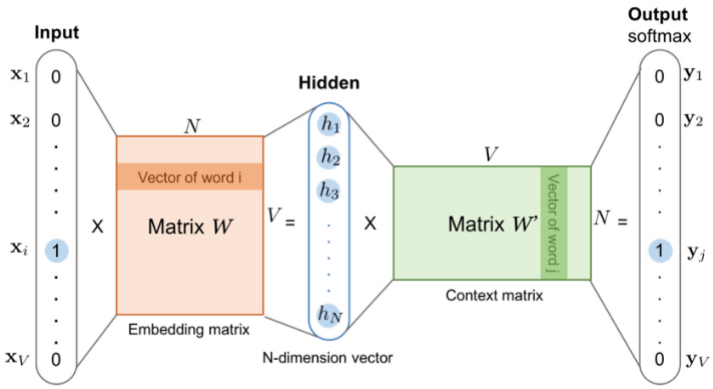
- Embedding是什么
  - Embedding 就是用一个数值向量“表示”一个对象（Object）的方法
  - 表示：一个物品能被向量表示，是因为这个向量跟其他物品向量之间的距离反映了这些物品的相似性。更进一步来说，两个向量间的距离向量甚至能够反映它们之间的关系
- Embedding 技术对深度学习推荐系统的重要性
  - Embedding 是处理稀疏特征的利器：几乎所有深度学习推荐模型都会由 Embedding 层负责将稀疏高维特征向量转换成稠密低维特征向量【特征过于稀疏会导致整个网络的收敛非常慢，因为每一个样本的学习只有极少数的权重会得到更新】
  - Embedding 可以融合大量有价值信息，本身就是极其重要的特征向量
- 经典的 Embedding 方法：Word2vec
  - Word2vec（“word to vector”）：生成对“词”的向量表达的模型；需要准备由一组句子组成的语料库。假设其中一个长度为 T 的句子包含的词有 w1,w2.....wt，并且我们假定每个词都跟其相邻词的关系最密切

    > CBOW模型：假设句子中每个词的选取都由相邻的词决定，输入是 wt周边的词，预测的输出是 wt
    >
    > Skip-gram模型：假设句子中的每个词都决定了相邻词的选取，输入是 wt，预测的输出是 wt周边的词；一般效果会更好

  - Word2vec 的训练样本生成过程

    > 
    >
    >
    > 通过滑动窗口一一截取词组，把词组内的词转换成训练样本

  - Word2vec 的模型结构：本质上是一个三层的神经网络

    > 
    >
    >
    > 输入层和输出层的维度都是V： V 是语料库词典的大小

> 隐层的维度是N：决定最终每个词的 Embedding 向量维度，其中embedding是输入层到隐层的权重矩阵
>
> 从 Word2vec 模型中提取词向量：在实际的使用过程中，我们往往会把输入向量矩阵转换成词向量查找表（Lookup table，如图 7 所示）。例如，输入向量是 10000 个词组成的 one hot 向量，隐层维度是 300 维，那么输入层到隐层的权重矩阵为 10000x300 维。在转换为词向量 Lookup table 后，每行的权重即成了对应词的 Embedding 向量。如果我们把这个查找表存储到线上的数据库中，就可以轻松地在推荐物品的过程中使用 Embedding 去计算相似性等重要的特征了。
>
> 激活函数：隐层神经元没有激活函数，输出层神经元使用 softmax

- Word2vec 的数学模型

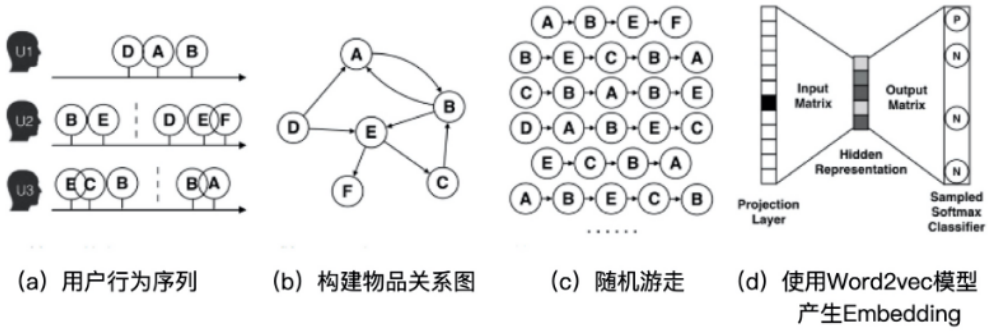
> $$p\left(w_O \mid w_I\right)=\frac{\exp\left(v_{w_O}^{\prime} v_{w_I}\right)}{\sum_{i=1}^{V} \exp\left(v_{w_i}^{\prime \top} v_{w_I}\right)}$$
>
> 这个由输入词 WI 预测输出词 WO 的条件概率，其实就是 Word2vec 神经网络要表达的东西。我们通过极大似然的方法去最大化这个条件概率，就能够让相似的词的内积距离更接近，这就是我们希望 Word2vec 神经网络学到的。

- Word2vec 值得挖掘的东西：为了节约训练时间，Word2vec 经常会采用负采样（Negative Sampling）或者分层 softmax（Hierarchical Softmax）的训练方法
- Word2vec 方法的推广：Item2Vec，序列数据

> 只要能够用序列数据的形式把我们要表达的对象表示出来，再把序列数据“喂”给 Word2vec 模型，我们就能够得到任意物品的 Embedding 了

Embedding进阶

- 图数据
  - 社交网络：意见领袖、社区
  - 知识图谱：包含了不同类型的知识主体（如人物、地点等），附着在知识主体上的属性（如人物描述，物品特点），以及主体和主体之间、主体和属性之间的关系
  - 行为关系类图数据：由用户和物品组成的“二部图”（也称二分图），用户和物品之间的相互行为生成了行为关系图
- 基于随机游走的 Graph Embedding 方法，Deep Walk
  - 主要思想：在由物品组成的图结构上进行随机游走，产生大量物品序列，然后将这些物品序列作为训练样本输入 Word2vec 进行训练，最终得到物品的 Embedding
  - 过程：

> 
>
> （a）用户行为序列　（b）构建物品关系图　（c）随机游走　（d）使用Word2vec模型产生Embedding
>
> 用户行为序列：因为用户 Ui先后购买了物品 A 和物品 B，所以产生了一条由 A 到 B 的有向边；如果后续产生了多条相同的有向边，则有向边的权重被加强
>
> 采用随机游走的方式随机选择起始点，重新产生物品序列

> 将这些随机游走生成的物品序列输入 Word2vec 模型，生成最终的物品 Embedding 向量
>
> 形式化定义：随机游走的跳转概率，也就是到达节点 vi后，下一步遍历 vi的邻接点 vj的概率
>
> $$P\left(v_j \mid v_i\right)=\left\{\begin{array}{ll} \frac{M_{ij}}{\sum_{j \in N_+\left(V_i\right)}}, m_{ij} & v_j \in N_+\left(v_i\right) \\ 0, & \mathrm{e\_}ij \notin \varepsilon \end{array}\right.$$

- 在同质性和结构性间权衡的方法，Node2vec
  - 通过调整随机游走跳转概率的方法，让 Graph Embedding 的结果在网络的同质性（Homophily）和结构性（Structural Equivalence）中进行权衡，可以进一步把不同的 Embedding 输入推荐模型，让推荐系统学习到不同的网络结构特点
  - “同质性”指的是距离相近节点的 Embedding 应该尽量近似：随机游走要更倾向于 DFS（Depth First Search，深度优先搜索）
  - “结构性”指的是结构上相似的节点的 Embedding 应该尽量接近：游走的过程更倾向于 BFS（Breadth First Search，宽度优先搜索）
  - 倾向于 DFS 还是 BFS：与跳转权重有关

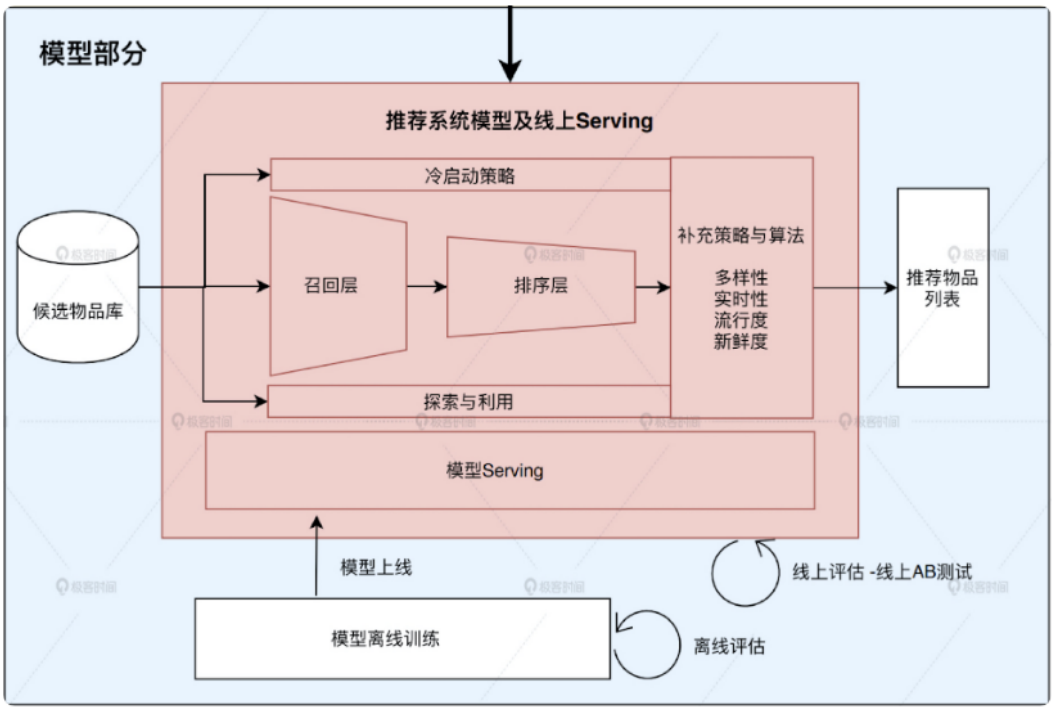
    > $$\alpha_{pq(t,x)=}\left\{\begin{array}{ll} \frac{1}{p} & \text{如果 } d_{tx}=0 \\ 1 & \text{如果 } d_{tx}=1 \\ \frac{1}{q} & \text{如果 } d_{tx}=2 \end{array}\right.$$
    >
    > d_{tx}是节点t到节点x的距离，直接相连的节点距离为1，节点到自己的距离是0
    >
    > p是返回参数：p越小，随机游走回节点t的可能性越大，Node2vec 就更注重表达网络的结构性
    >
    > q是进出参数：q 越小，随机游走到远方节点的可能性越大，Node2vec 更注重表达网络的同质性

- Embedding在推荐系统的应用
  - 直接应用：在得到 Embedding 向量之后，直接利用 Embedding 向量的相似性实现某些推荐系统的功能
  - 预训练应用：把这些 Embedding 向量作为特征向量的一部分，跟其余的特征向量拼接起来，作为推荐模型的输入参与训练
  - End to end training：我们不预先训练 Embedding，而是把 Embedding 的训练与深度学习推荐模型结合起来，采用统一的、端到端的方式一起训练，直接得到包含 Embedding 层的推荐模型（微软的 Deep Crossing，UCL 提出的 FNN 和 Google 的 Wide&Deep）

线上服务篇

线上服务：用 Jetty Server 搭建推荐服务器

- 工业级推荐服务器的功能



高并发推荐服务的整体架构主要由三个重要机制支撑，它们分别是负载均衡、缓存、推荐服务降级机制

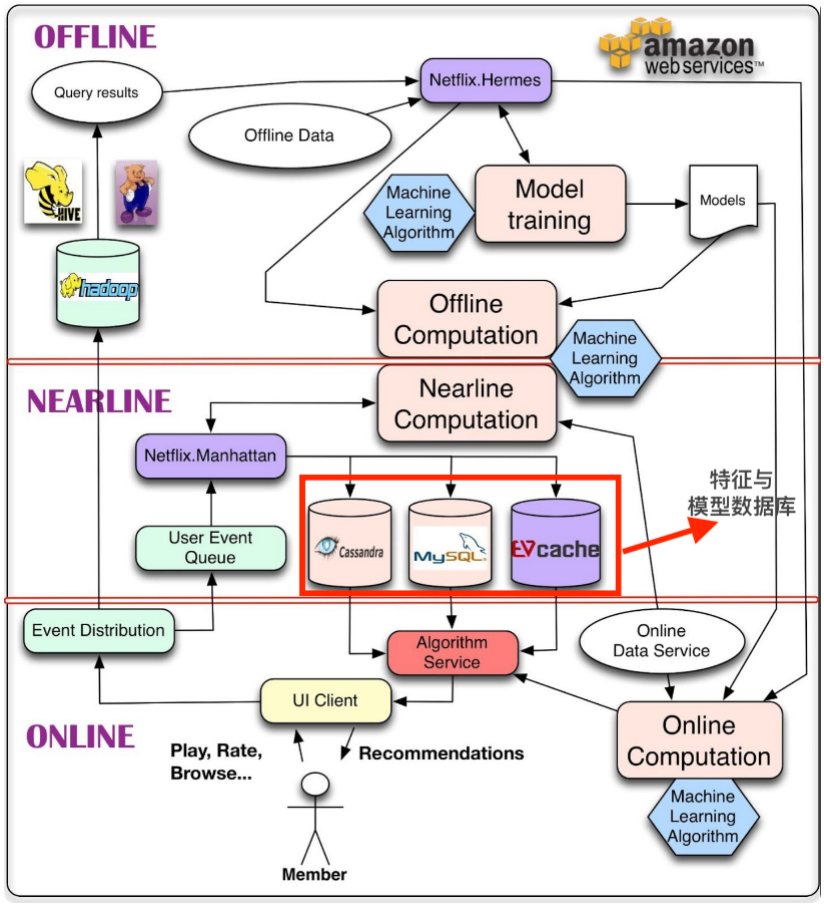
- 负载均衡：需要增加服务器来分担独立节点的压力，负载均衡服务器负责分配任务，以达到按能力分配和高效率分配的目的；nginx技术选型
- 缓存：通过“减少工作量”解决高并发带来的负载压力；当同一个用户多次请求同样的推荐服务时，我们就可以在第一次请求时把 TA 的推荐结果缓存起来，在后续请求时直接返回缓存中的结果；再比如说，对于新用户来说，因为他们几乎没有行为历史的记录，所以我们可以先按照一些规则预先缓存好几类新用户的推荐列表，等遇到新用户的时候就直接返回
- 服务降级：抛弃原本的复杂逻辑，采用最保险、最简单、最不消耗资源的降级服务来渡过特殊时期

- SparrowRecys选择的服务器框架：Java 嵌入式服务器 Jetty

存储模块：用 Redis 解决特征存储的问题

- 推荐系统存储模块的设计原则

> Netflix的Offline、Nearline、Online 三层推荐系统架构



Cassandra、MySQL 和 EVcache这三个数据库是 Netflix 解决特征和模型参数存储问题的关键

- Cassandra：流行的 NoSQL 数据库，具备大数据存储的能力
- EVcache：内存数据库，保存最常用的特征和模型参数，支持服务器高QPS需求
- MySQL：强一致性的关系型数据库，一般存储的是比较关键的要求强一致性的信息，比如物品是否可以被推荐这种控制类的信息，物品分类的层级关系，用户的注册信息等等。这类信息一般是由推荐服务器进行阶段性的拉取，或者利用分级缓存进行阶段性的更新，避免因为过于频繁的访问压垮 MySQL

- 难点：又要存储量大，又要查询快，还要面对高 QPS 的压力
- 工业级推荐系统设计原则：把特征的存储做成分级存储，把越频繁访问的数据放到越快的数据库甚至缓存中，把海量的全量数据放到便宜但是查询速度较慢的数据库中

- SparrowRecsys的存储系统方案
  - 问题简化：使用基础的文件系统保存全量的离线特征和模型数据，用 Redis 保存线上所需特征和模型数据，使用服务器内存缓存频繁访问的特征
  - 特征和模型数据

特征类型	具体特征	数据量级
用户特征	用户的评分历史等	百万级别（样例数据中是万级别）
物品特征	电影的平均分，发布年份，电影类型等	万级别（样例数据中是千级别）
用户Embedding	使用不同Embedding方法生成的用户Embedding	百万级别（样例数据中是万级别）
物品Embedding	使用不同Embedding方法生成的物品Embedding	万级别（样例数据中是千级别）

用户特征的总数比较大，它们很难全部载入到服务器内存中，所以我们把用户特征载入到 Redis 之类的内存数据库中是合理的

> 物品特征的总数比较小，而且每次用户请求，一般只会用到一个用户的特征，但为了物品排序，推荐服务器需要访问几乎所有候选物品的特征。针对这个特点，我们完全可以把所有物品特征阶段性地载入到服务器内存中，大大减少 Redis 的线上压力
>
> 由于类似 HDFS 之类的分布式文件系统具有近乎无限的存储空间，我们可以把每次处理的全量特征，每次训练的 Embedding 全部保存到分布式文件系统中，方便离线评估时使用。

- 最终的存储方案

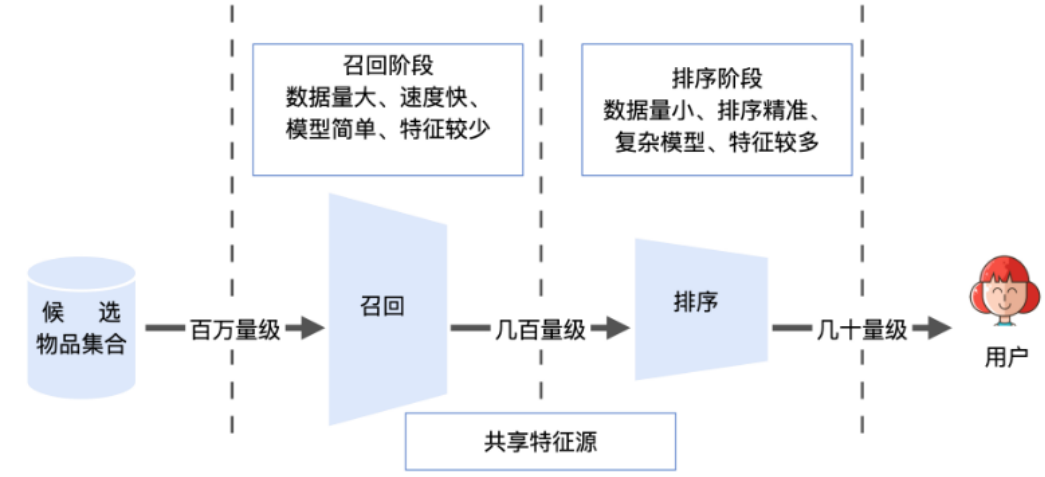
> | 存储系统类型 | 存储特征 |
> | --- | --- |
> | 分布式文件系统 | 历次处理得出的全量特征 |
> | 内存数据库（Redis） | 当前版本的所有用户、物品特征 |
> | 服务器内存 | 物品特征阶段性全量载入，
用户特征根据请求中的用户ID实时请求Redis |
>
> 文件系统的存储操作在 SparrowRecsys 中利用 Spark 的输出功能实现，服务器内部的存储操作主要是跟 Redis 进行交互

- Redis基础
  - 两个主要特点：
    1. 所有的数据都以 Key-value 的形式存储：Key 只能是字符串，value 可支持的数据结构包括 string(字符串)、list(链表)、set(集合)、zset(有序集合) 和 hash(哈希)
    2. 所有的数据都存储在内存中，磁盘只在持久化备份或恢复数据时起作用：Redis 的特性一是 QPS 峰值可以很高，二是数据易丢失；对于可恢复，不关乎关键业务逻辑的推荐特征数据，就非常适合利用 Redis 提供高效的存储和查询服务
  - 实践流程
    1. 安装 Redis
    2. 运行离线程序，通过 jedis 客户端写入 Redis：利用最常用的 Redis Java 客户端 Jedis 生成 redisClient，然后遍历训练好的 Embedding 向量，将 Embedding 向量以字符串的形式存入 Redis，并设置过期时间（ttl）

       代码参考 com.wzhe.sparrowrecsys.offline.spark.featureeng.Embedding 中的 trainItem2vec 函数
    3. 在推荐服务器中把 Redis 数据读取出来
  - 安装好Redis后，运行 SparrowRecsys 中 offline 部分 Embedding 主函数，先把物品和用户 Embedding 生成并且插入 Redis（注意把 saveToRedis 变量改为 true）。然后再运行 online 部分的 RecSysServer，看一下推荐服务器有没有正确地从 Redis 中读出物品和用户 Embedding 并产生正确的推荐结果

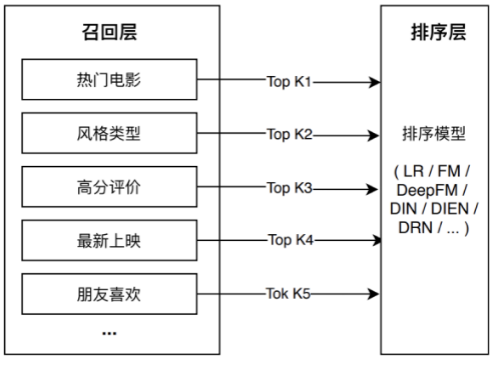
召回层

- 召回层和排序层的功能特点



从技术架构的角度来说，“召回层”处于推荐系统的线上服务模块之中，推荐服务器从数据库或内存中拿到所有候选物品集合后，会依次经过召回层、排序层、再排序层（也被称为补充算法层），才能够产生用户最终看到的推荐列表

    - 召回层设计：计算速度和召回率其实是两个矛盾的指标；为了提高计算速度，我们需要使召回策略尽量简单，而为了提高召回率或者说召回精度，让召回策略尽量把用户感兴趣的物品囊括在内，这又要求召回策略不能过于简单
- 单策略召回
    - 通过制定一条规则或者利用一个简单模型来快速地召回可能的相关物品，这里的规则其实就是用户可能感兴趣的物品的特点
    - 在推荐电影的时候，我们首先要想到用户可能会喜欢什么电影。按照经验来说，很有可能是这三类，分别是大众口碑好的、近期非常火热的，以及跟我之前喜欢的电影风格类似的，基于其中任何一条，我们都可以快速实现一个单策略召回层
    - 它有很强的局限性。因为大多数时候用户的兴趣是非常多元的，他们不仅喜欢自己感兴趣的，也喜欢热门的，当然很多时候也喜欢新上映的
    - SparrowRecSys：candidateGenerator
- 多路召回
    - 采用不同的策略、特征或简单模型，分别召回一部分候选集，然后把候选集混合在一起供后续排序模型使用的策略
    - 具体过程：



    - SparrowRecsys：实现了由风格类型、高分评价、最新上映，这三路召回策略组成的多路召回方法multipleRetrievalCandidates
    - 在实现的过程中，为了进一步优化召回效率，我们还可以通过多线程并行、建立标签 / 特征索引、建立常用召回集缓存等方法来进一步完善它

    - 缺点：在确定每一路的召回物品数量时，往往需要大量的人工参与和调整，具体的数值需要经过大量线上 AB 测试来决定。此外，因为策略之间的信息和数据是割裂的，所以我们很难综合考虑不同策略对一个物品的影响
- 基于embedding的召回方法：利用物品和用户 Embedding 相似性来构建召回层
    - 多路召回中使用的“兴趣标签”“热门度”“流行趋势”“物品属性”等信息都可以作为 Embedding 方法中的附加信息（Side Information），融合进最终的 Embedding 向量中。因此，在利用 Embedding 召回的过程中，我们就相当于考虑到了多路召回的多种策略。
    - Embedding 召回的评分具有连续性。我们知道，多路召回中不同召回策略产生的相似度、热度等分值不具备可比性，所以我们无法据此来决定每个召回策略放回候选集的大小。但是，Embedding 召回却可以把 Embedding 间的相似度作为唯一的判断标准，因此它可以随意限定召回的候选集大小。
    - 在线上服务的过程中，Embedding 相似性的计算也相对简单和直接。通过简单的点积或余弦相似度的运算就能够得到相似度得分，便于线上的快速召回。
    - SparrowRecsys也实现了基于embedding的召回方法：retrievalCandidatesByEmbedding
    - 整体思路
        - 第一步，我们获取用户的 Embedding
        - 第二步，我们获取所有物品的候选集，并且逐一获取物品的 Embedding，计算物品 Embedding 和用户 Embedding 的相似度【主要时间开销】
        - 第三步，我们根据相似度排序，返回规定大小的候选集

局部敏感哈希：在常数时间内搜索embedding最近邻

- embedding最近邻搜索问题
    - 假设用户和物品的 Embeding 都在一个 k 维的 Embedding 空间中，物品总数为 n，那么遍历计算一个用户和所有物品向量相似度的时间复杂度是$O(k \times n)$
    - 召回与用户向量最相似的物品 Embedding 向量这一问题，其实就是在向量空间内搜索最近邻的过程
- 使用“聚类”还是“索引”来搜索最近邻?

    > 聚类通过把相似的点聚类到一起，快速地找到彼此间的最近邻
    >
    > 索引通过某种数据结构建立基于向量距离的索引，在查找最近邻的时候，通过索引快速缩小范围来降低复杂度

    - 聚类方法：K-means
        - 在离线计算好每个 Embedding 向量的类别，在线上我们只需要在同一个类别内的 Embedding 向量中搜索就可以了
        - 局限性：聚类边缘的点的最近邻往往会包括相邻聚类的点，如果我们只在类别内搜索，就会遗漏这些近似点。此外，中心点的数量 k 也不那么好确定，k 选的太大，离线迭代的过程就会非常慢，k 选的太小，在线搜索的范围还是很大，并没有减少太多搜索时间
    - 索引方法：kd-tree
        - 离线建好索引后，把索引搬到线上，利用二叉树的结构快速找到邻接点
        - 局限性：无法完全解决边缘点最近邻的问题；kd-tree 索引的结构并不简单，离线和在线维护的过程也相对复杂
- 局部敏感哈希的基本原理及多桶策略
    - 基本思想：希望让相邻的点落入同一个“桶”
    - 利用低维空间可以保留高维空间相近距离关系的性质构造局部敏感哈希“桶”

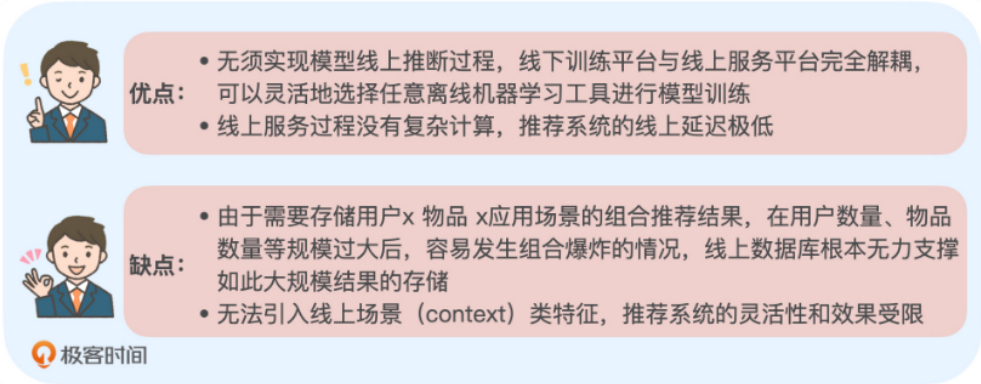
        > 欧式空间中，将高维空间的点映射到低维空间，原本接近的点在低维空间中肯定依然接近，但原本远离的点则有一定概率变成接近的点

> 假设 v 是高维空间中的 k 维 Embedding 向量，x 是随机生成的 k 维映射向量。那我们利用内积操作可以将 v 映射到一维空间，得到数值 h(v)=v·x
>
> 因为一维空间会部分保存高维空间的近似距离信息，我们可以使用哈希函数 h(v) 进行分桶，公式为：$h^{x,b} = \lfloor \frac{x \cdot v + b}{w} \rfloor$（w是分桶宽度，b是0到w间的一个均匀分布随机变量，避免分桶边界固话）
>
> 映射操作会损失部分距离信息，如果我们仅采用一个哈希函数进行分桶，必然存在相近点误判的情况，因此，我们可以采用 m 个哈希函数同时进行分桶。如果两个点同时掉进了 m 个桶，那它们是相似点的概率将大大增加
>
> 通过分桶找到相邻点的候选集合后，我们就可以在有限的候选集合中通过遍历找到目标点真正的 K 近邻了

    - 多桶策略：如果有多个分桶函数的话，具体应该如何处理不同桶之间的关系
        - 且操作：可以最大程度地减少候选点数量，但是会增大漏掉最近邻点的概率
        - 或操作：增大了候选集的规模，减少了漏掉最近邻点的可能性，但增大了后续计算的开销
    - 工程经验
        - 点数越多，我们越应该增加每个分桶函数中桶的个数；相反，点数越少，我们越应该减少桶的个数
        - Embedding 向量的维度越大，我们越应该增加哈希函数的数量，尽量采用且的方式作为多桶策略；相反，Embedding 向量维度越小，我们越应该减少哈希函数的数量，多采用或的方式作为分桶策略
- 局部敏感哈希实践：embeddingLSH
- 在将电影 Embedding 数据转换成 dense Vector 的形式之后，我们使用 Spark MLlib 自带的 LSH 分桶模型 BucketedRandomProjectionLSH（我们简称 LSH 模型）来进行 LSH 分桶。其中最关键的部分是设定 LSH 模型中的 BucketLength 和 NumHashTables 这两个参数。其中，BucketLength 指的就是公式 2 中的分桶宽度 w，NumHashTables 指的是多桶策略中的分桶次数。
- 超大规模最近邻搜索问题的业界解决方案：Facebook的开源向量最近邻搜索库FAISS（https://github.com/facebookresearch/faiss）

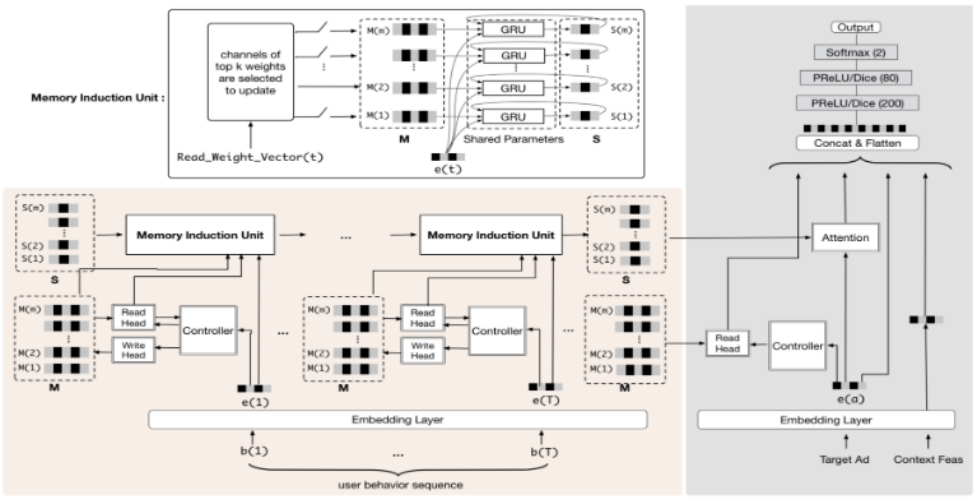
模型服务：把离线模型部署到线上

- 预存推荐结果或 Embedding 结果
    - 在离线环境下生成对每个用户的推荐结果，然后将结果预存到以 Redis 为代表的线上数据库中。这样，我们在线上环境直接取出预存数据推荐给用户即可
    - 优缺点



- 预训练 Embedding+ 轻量级线上模型

- 用复杂深度学习网络离线训练生成 Embedding，存入内存数据库，再在线上实现逻辑回归或浅层神经网络等轻量级模型来拟合优化目标
- 阿里的推荐模型 MIMN（Multi-channel user Interest Memory Network，多通道用户兴趣记忆网络）



线下：左边巨复杂的部分

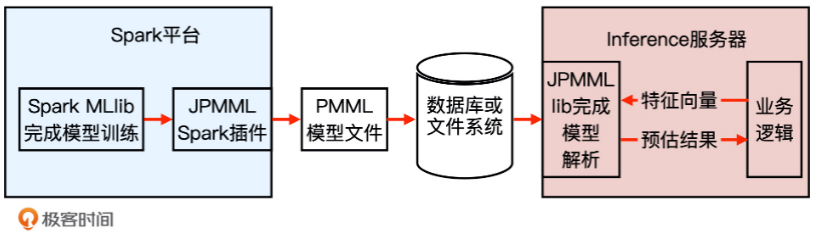
线上：右边灰色的神经网络

接口：图中连接处的位置有两个被虚线框框住的数据结构，分别是 S(1)-S(m) 和 M(1)-M(m)。它们其实就是在离线生成的 Embedding 向量，在 MIMN 模型中，它们被称为“多通道用户兴趣向量”

工作过程：线上部分从 Redis 之类的模型数据库中拿到这些离线生成 Embedding 向量，然后跟其他特征的 Embedding 向量组合在一起，扔给一个标准的多层神经网络进行预估

- 隔离了离线模型的复杂性和线上推断的效率要求，但还是造成了模型的割裂

- 利用 PMML 转换和部署模型
  - PMML 的全称是“预测模型标记语言”(Predictive Model Markup Language, PMML)，它是一种通用的以 XML 的形式表示不同模型结构参数的标记语言。在模型上线的过程中，PMML 经常作为中间媒介连接离线训练平台和线上预测平台
  - Spark模型利用PMML的上线过程



使用了 JPMML 作为序列化和解析 PMML 文件的 library（库）

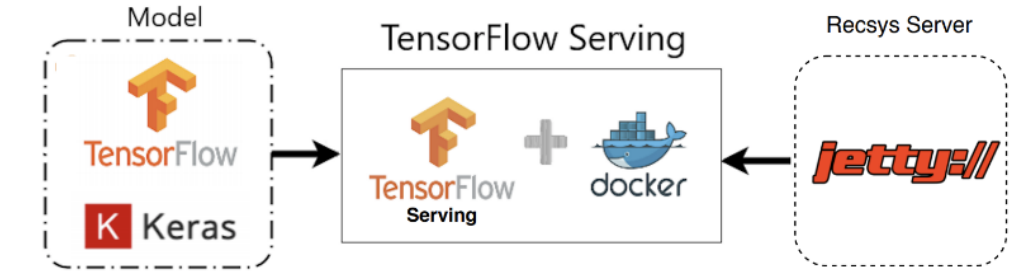
JPMML 项目分为 Spark 和 Java Server 两部分：

- Spark 部分的 library 完成 Spark MLlib 模型的序列化，生成 PMML 文件，并且把它保存到线上服务器能够触达的数据库或文件系统中
- Java Server 部分则完成 PMML 模型的解析，生成预估模型，完成了与业务逻辑的整合

JPMML的项目地址：https://github.com/jpmml

  - PMML 语言的表示能力还是比较有限的，还不足以支持复杂的深度学习模型结构

- TensorFlow Serving
  - 从整体工作流程来看，TensorFlow Serving 和 PMML 类工具的流程一致，它们都经历了模型存储、模型载入还原以及提供服务的过程
  - 在具体细节上，TensorFlow 在离线把模型序列化，存储到文件系统，TensorFlow Serving 把模型文件载入到模型服务器，还原模型推断过程，对外以 HTTP 接口或 gRPC 接口的方式提供模型服务
  - SparrowRecsys 项目

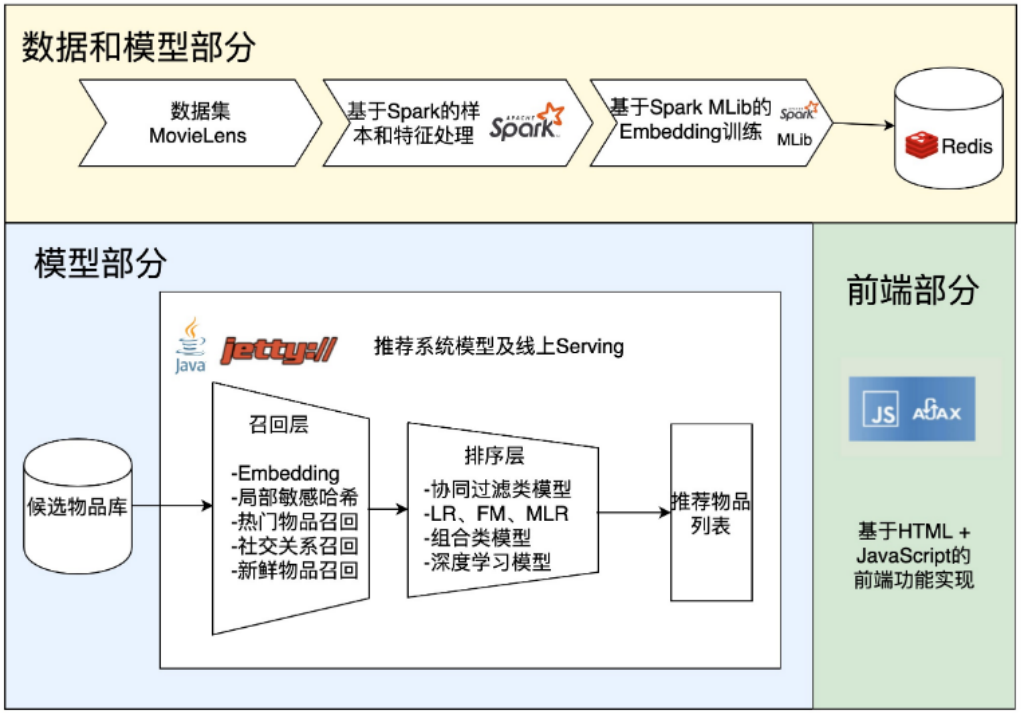


离线使用 TensorFlow 的 Keras 接口完成模型构建和训练，再利用 TensorFlow Serving 载入模型，用 Docker 作为服务容器，然后在 Jetty 推荐服务器中发出 HTTP 请求到 TensorFlow Serving，获得模型推断结果，最后推荐服务器利用这一结果完成推荐排序

- 实战搭建 TensorFlow Serving 模型服务
  1. 安装Docker：开源的应用容器引擎，类似于一个轻量级的虚拟机；通过使用 Docker 建立模型服务 API
  2. 建立TensorFlow Serving 服务
  3. 请求 TensorFlow Serving 获得预估结果

在SparrowRecsys中实现相似电影推荐功能

- 技术架构图：精简了深度学习推荐模型和模型评估部分



- 数据和模型部分
  - 采用 Spark 进行数据处理，同时选择 Spark MLlib 进行 Embedding 的训练。这部分内容的代码，你可以参考项目中的

_com.wzhe.sparrowrecsys.offline.spark.embedding.__Embedding_对象，它定义了所有项目中用到的 Embedding 方法
    - 为了方便线上服务使用，我们还需要在生成 Embedding 后，把它们存入某个高可用的数据库。SparrowRecsys 选择了最主流的内存数据库 Redis 作为实现方案，这一部分的具体实现，你可以参照 com.wzhe.sparrowrecsys.offline.spark.embedding.Embedding 对象中 trainItem2vec 函数的 Redis 存储操作
- 线上服务部分：候选物品库的建立、召回层的实现、排序层的实现
    - 候选物品库的建立
        - 项目是直接把 MovieLens 数据集中的物品数据载入到内存中
        - 业级推荐系统往往会通过比较复杂的 SQL 查询，或者 API 查询来获取候选集
    - 召回层的实现
        - 使用 Embedding 召回的方法来完成召回层的实现
        - SparrowRecsys 也实现了基于物品 metadata（元信息）的多路召回方法，具体的实现你可以参照 com.wzhe.sparrowrecsys.online.recprocess.SimilarMovieProcess 类中的 multipleRetrievalCandidates 函数和 retrievalCandidatesByEmbedding 函数
    - 排序层的实现
        - 先根据召回层过滤出候选集，再从 Redis 中取出相应的 Embedding 向量，然后计算目标物品和候选物品之间的相似度，最后进行排序就可以了
        - 因为在 Word2vec 中，相似度的定义是内积相似度，所以, 这里我们也采用内积作为相似度的计算方法。具体的实现可以参照 com.wzhe.sparrowrecsys.online.recprocess.SimilarMovieProcess 类中的 ranker 函数
- 前端部分：HTML+AJAX 请求
    - AJAX 的全称是 Asynchronous JavaScript and XML，异步 JavaScript 和 XML 请求
    - AJAX指的是不刷新整体页面，用 JavaScript 异步请求服务器端，更新页面中部分元素的技术。当前流行的 JavaScript 前端框架 React、Vue 等等也大多是基于 AJAX 来进行数据交互的
- 相似电影推荐的结果和初步分析
    - 人肉测试（SpotCheck）
    - 指定 Ground truth（可以理解为标准答案）
    - 利用商业指标进行评估：跃过评估相似度这样一个过程，直接去评估它的终极商业指标

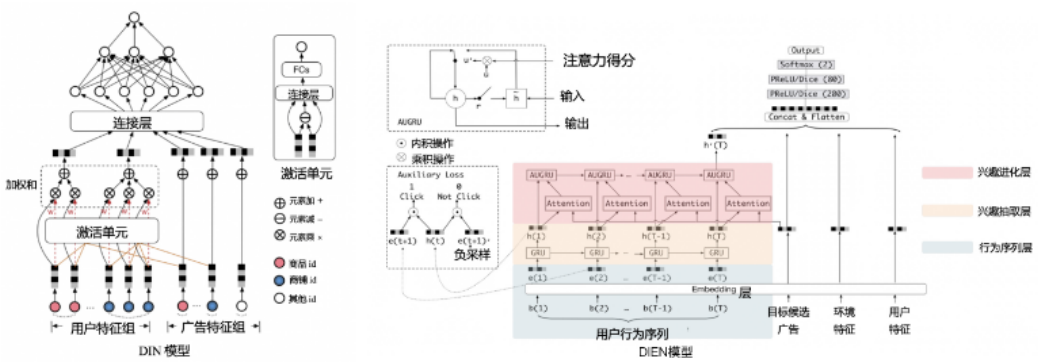
推荐模型篇

协同过滤

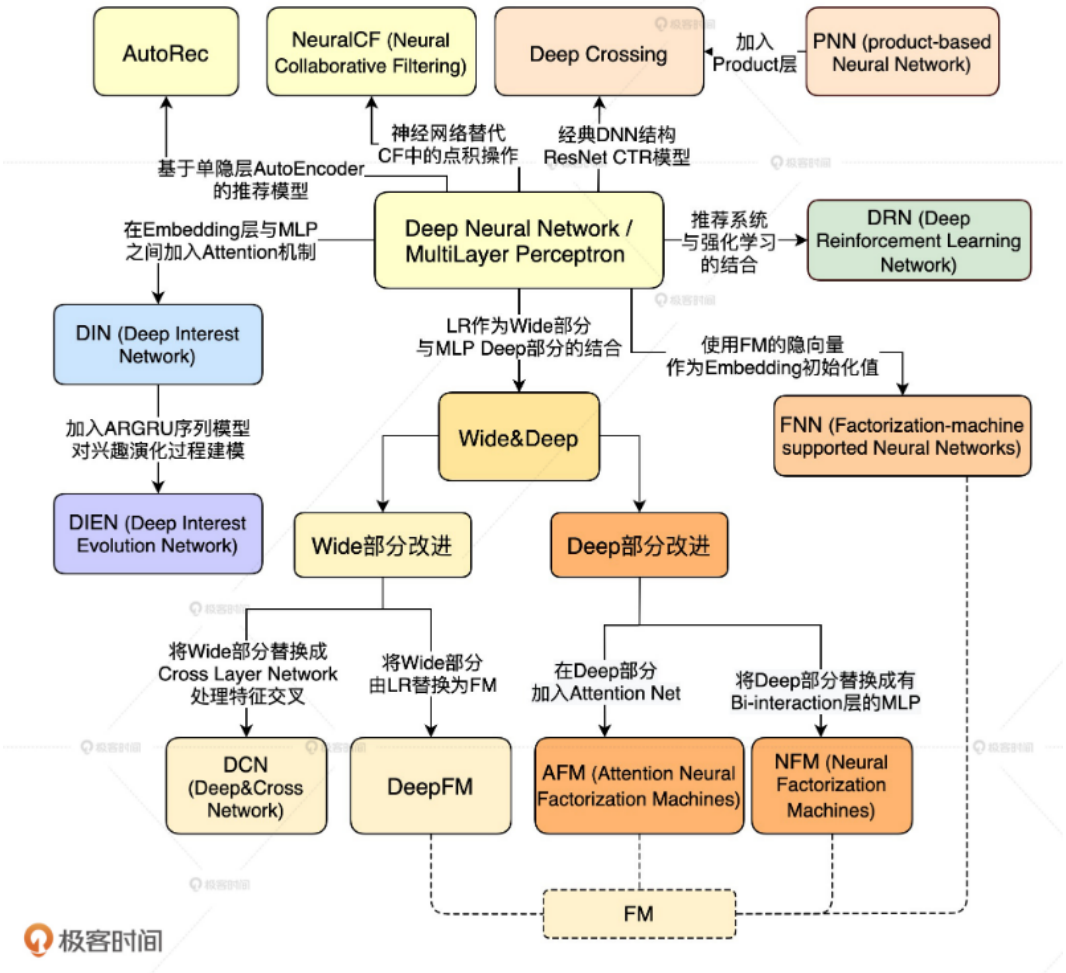
- 协同过滤算法：让用户考虑与自己兴趣相似用户的意见
    - 共现矩阵
    - 计算用户相似度：余弦相似度
    - 用户评分的预测：在获得 Top n 个相似用户之后，利用 Top n 用户生成最终的用户 u 对物品 p 的评分
    - 缺点：共现矩阵往往非常稀疏，在用户历史行为很少的情况下，寻找相似用户的过程并不准确
- 矩阵分解算法：加强了模型处理稀疏矩阵的能力
    - 相当于一种 Embedding 方法：先分解协同过滤生成的共现矩阵，生成用户和物品的隐向量，再通过用户和物品隐向量间的相似性进行推荐
    - 把共现矩阵分解开的方法：梯度下降，通过求取偏导的形式来更新权重
- 矩阵分解算法的Spark实现
- 在 SparrowRecSys 中线上部署：矩阵分解算法得出的结果可以当作 Embedding 来处理

深度学习推荐模型发展的整体脉络

- 深度学习模型的优势：强拟合能力、结构的灵活性

> 阿里巴巴DIN模型（深度兴趣网络）和 DIEN（深度兴趣进化网络）
>
> 
>
>
> 图4 DIN模型（左）和DIEN的模型（右）示意图
>
> DIN 模型：在神经网络中增加了一个叫做“激活单元”的结构，这个单元就是为了模拟人类的注意力机制。举个例子来说，我们在购买电子产品，比如说笔记本电脑的时候，更容易拿之前购买电脑的经验，或者其他电子产品的经验来指导当前的购买行为，很少会借鉴购买衣服和鞋子的经验。这就是一个典型的注意力机制，我们只会注意到相关度更高的历史购买行为
>
> DIEN模型：不仅引入了注意力机制，还模拟了用户兴趣随时间的演化过程。我们来看那些彩色的层，这一层层的序列结构模拟的正是用户兴趣变迁的历史，通过模拟变迁的历史，DIEN模型可以更好地预测下一步用户会喜欢什么

- 深度学习推荐模型的演化关系图

> 
>

> MLP是整个演化图的核心：一个黑盒，对输入的特征进行深度地组合交叉，然后输出对兴趣值的预测
>
> Deep Crossing：在原始特征和 MLP 之间加入了 Embedding 层。这样一来，输入的稀疏特征先转换成稠密 Embedding 向量，再参与到 MLP 中进行训练，这就解决了 MLP 不善于处理稀疏特征的问题
>
> Wide&Deep：把深层的 MLP 和单层的神经网络结合起来，希望同时让网络具备很好的“记忆性”和“泛化性”
>
> 深度学习和注意力机制的结合：阿里的深度兴趣网络 DIN，浙大和新加坡国立提出的 AFM 等等
>
> 把序列模型引入 MLP+Embedding 的经典结构：阿里的深度兴趣进化网络 DIEN
>
> 把深度学习和强化学习结合在一起：微软的深度强化学习网络 DRN，以及包括美团（https://tech.meituan.com/2018/11/15/reinforcement-learning-in-mt-recommend-system.html）、阿里在内的非常有价值的业界应用

- 模型改进的4个方向
  - 改变神经网络的复杂程度：从最简单的单层神经网络模型 AutoRec，到经典的深度神经网络结构 Deep Crossing，它们主要的进化方式在于增加了深度神经网络的层数和结构复杂度
  - 改变特征交叉方式：这种演进方式的要点在于大大提高了深度学习网络中特征交叉的能力。比如说，改变了用户向量和物品向量互操作方式的 NeuralCF，定义了多种特征向量交叉操作的 PNN 等等
  - 把多种模型组合应用：以 Wide&Deep 模型为代表的一系列把不同结构组合在一起的改进思路。它通过组合两种甚至多种不同特点、优势互补的深度学习网络，来提升模型的综合能力
  - 让深度推荐模型和其他领域进行交叉：一般来说，自然语言处理、图像处理、强化学习这些领域都是推荐系统经常汲取新知识的地方