综合实验： 实现推荐系统

实验目的：

（1）掌握 Lambda 架构

实验平台：

操作系统：Ubuntu 18.04 LTS

实验内容：

（1）基于 Lambda 架构的大数据推荐系统架构概述

Lambda Architecture（LA）最早是 Twitter 工程师 Nathan Marz 提出来的，它是一种大数据软件设计架构，其目的是指导用户充分利用批处理和流式计算任务各自的优点实现一个复杂的大数据处理系统。通过结合这两类计算技术，LA 可以在延迟、吞吐量和容错之间找到平衡。

Lambda 架构的一个典型应用是推荐系统。在互联网行业，推荐系统被应用在各个领域，包括电子商务、视频、新闻等。推荐系统的设计目的是根据用户的兴趣特点和购买行为，向用户推荐感兴趣的信息和商品。推荐系统是建立在海量数据挖掘上的一种高级商务智能平台，以帮助商家为顾客购物提供完全个性化的决策支持和信息服务。

下图为一个最小化的推荐系统原型。在该原型中，用户产生的数据会被 Flume 收集，经 Kafka 的传输，被流式计算任务（如 Spark Streaming 等）和存储组件（HBase）消费。批式处理与流式处理会利用用户产生的数据计算特征与训练预测模型，两者不同点在于批处理利用 HBase 中存储的数据（通常为某个时间点之前的所有历史数据），流式处理利用 Kafka 中最近一段时间内的数据。最后，得到的计算特征与预测模型等都会存储在 Redis 数据库中，供服务端使用。服务端在接受到客户端的访问服务请求后，会根据 Redis 存储的特征与预测模型参数，计算最符合用户请求的商品列表，并返回客户端。客户端再根据用户的行为反馈（比如点击/不点击）向 Flume 发送日志，形成闭环。同时，在系统后台可以利用 Hive/Spark SQL 对 HBase 中的数据进行监控，并进一步完成数据的分析。

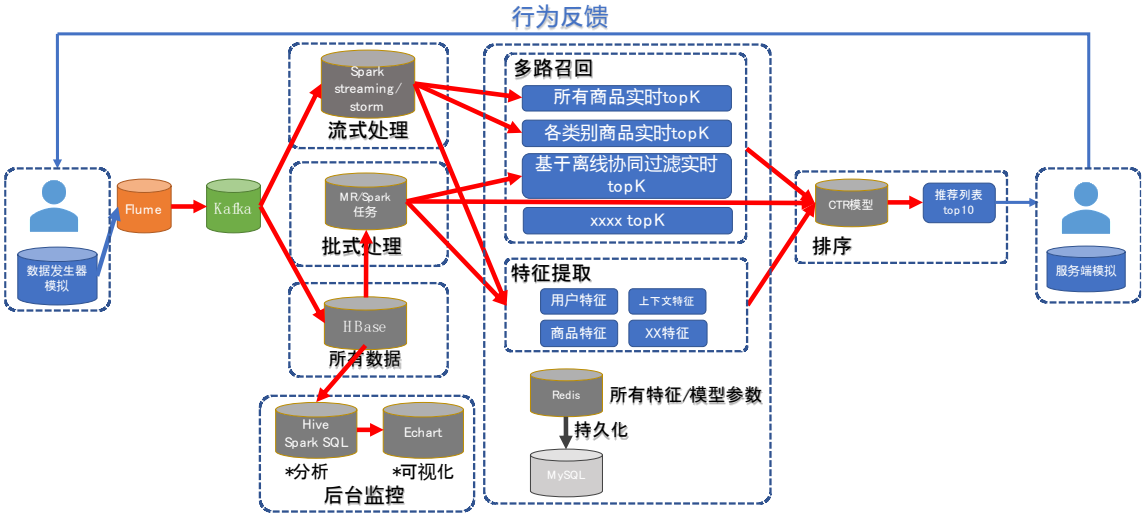


图 1 基于 Lambda 架构的推荐系统

推荐系统的核心是推荐算法，推荐算法通常会根据用户的兴趣特点和历史行为数据构建推荐模型，以预测用户可能感兴趣的信息和商品，进而推荐给用户。推荐算法

通常可以划分为两个阶段①召回阶段②排序阶段，如下图 2 所示。召回阶段就是从所有商品中，粗筛选出一些候选的商品，例如根据用户的兴趣标签 TopK、社区热门商品 TopK、某个算法模型的推荐列表 TopK（比如实验四中提到的协同过滤模型）。从多种途径获取候选列表也称为多路召回。排序阶段就是根据召回阶段粗筛选出来的候选商品，将之前批式处理、流式处理计算好的特征输入到训练好的点击率预测模型（也称为 CTR 模型）中，得到用户点击每个候选商品的概率并进行排序，最后将点击率最高 TopK 个商品返回给客户端。

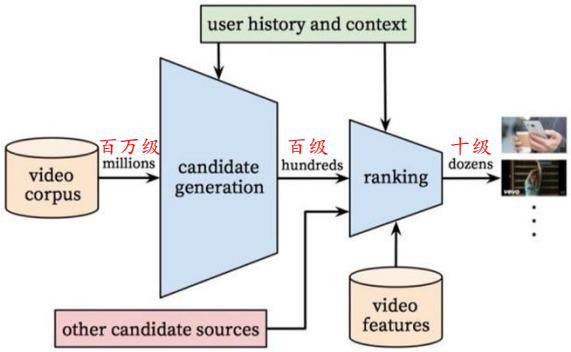


图 2 一个视频推荐算法流程（图片来自网络）

下面将分别对每个组件的所需要完成的功能进行讲解。

（2）使用 Flume、Kafka 实现数据的收集

启动一个 Flume 服务器进行数据收集。本次实验要求 Flume 服务器**以 HTTP Source 作为收集端**。本次收集的数据格式在附加说明（2）部分。启动一个 Kafka 服务器，并创建名为“movie_rating_records”的 Topic。Flume 将收集到的用户行为传输到 Kafka “movie_rating_records” Topic 下。

（3）使用 HBase 实现原始数据的存储

在 HBase 中创建一个表并命名为“movie_records”，数据格式在附加说明（3）部分。编写一个 Kafka 消费者，定期从“movie_rating_records” Topic 中获取数据并写入 HBase。

（4）使用 Redis 作为缓存数据库实现特征/模型参数的存储

Redis 是一个开源的、可基于内存也可持久化的 Key-Value 的 NoSQL 数据库。其性能极高，读写速度可达 10W/qps。同时，Redis 在 2015 年的 redis3.0+版本中加入了 redis cluster，可以实现分布式集群的架构，将原本单机的数据进行分区，分到若干个子集中。

Redis 不仅可以实现高效读写，还可以控制数据的过期时间，因此可被用于大数据组件之间中间结果的缓存。在本次实验中，将使用 Redis 作为批式、流式、服务等组件之间的数据交换中间件。例如，将批式计算的得到的特征写入 Redis 中，等待模型训练时再从 Redis 读取特征。

（5）使用 MapReduce 或 Spark 实现定时启动的批式计算任务

使用 MapReduce/Spark 启动定时任务，定时时间为 5 分钟(5*60*1000ms)，计算以下两部分内容：

①历史特征的计算

从 HBase 中读取所有历史数据，并计算以下五个批式特征，具体如下表所示。

特征	说明
batch2feature_userId_rating1_{userId}	用户历史正反馈次数
batch2feature_userId_rating0_{userId}	用户历史负反馈次数

batch2feature_userId_to_genresId_{userId}_{genresId}	用户历史点击该分类比例
batch2feature_movieId_rating1_{movieId}	电影历史被正反馈次数
batch2feature_movieId_rating0_{movieId}	电影历史被负反馈次数

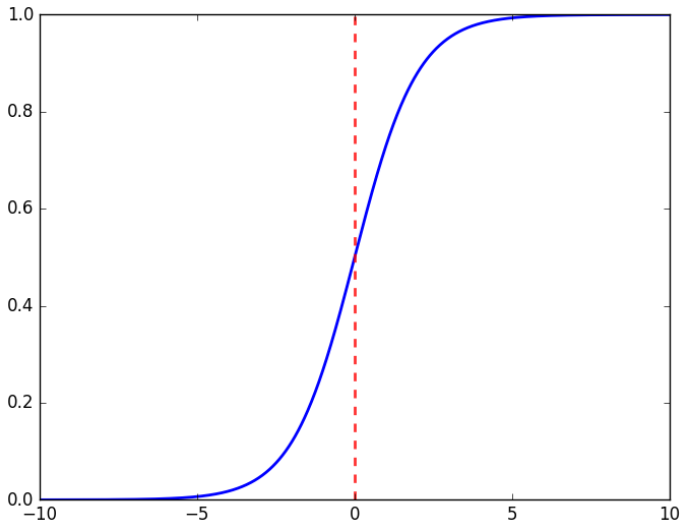
②CTR 预测模型的训练

Logistic Regression 算法是机器学习常用的有监督学习算法。可以将 Spark MLLib 中的 Logistic Regression 模型作为本次实验的 CTR 模型。

线性回归是监督学习中最简单的模型之一，该模型通过训练一条直线来拟合样本数据。其公式如下：

$$f(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n + b$$

其中，x_i表示样本的第 i 个特征，θ_i表示该特征在模型中的参数，b 表示偏置值，$f(x)$表示模型对于该样本的预测值。在训练阶段，通过已知的样本特征x_i和真实值y_i计算θ_i与 b。在预测阶段模，对于未知真实值的样本，使用模型参数θ_i、b 与样本特征x_i计算$f(x)$预估样本的真实值。但是，线性回归只能做连续值的预测，没办法完成二分类任务（比如 CTR 模型中的点击与不点击就是一个二分类问题）。要解决这个问题只需要找一个单调可微的函数将线性预测的连续值映射成 1/0 两个离散值。例如对数机率函数$S(x) = \frac{1}{1+e^{-x}}$（logistic function），也称为 sigmoid 函数。它的函数图像如下：



将线性回归的预测值$f(x)$再经过 sigmoid 函数，即$F(x) = \frac{1}{1+e^{-f(x)}} = \frac{1}{1+e^{-(\theta_0 x_0+\theta_1 x_1+\cdots+\theta_n x_n+b)}}$，就变成逻辑回归。

Spark MLlib 中的 spark.ml.LogisticRegression 类接收的训练数据的形式为一个 DataFrame，其中包括两列：

label: DoubleType（表示数据的标签）

features: VectorUDT（表示数据的特征）

如下图所示

```
+-----+--------------------------------------------------------------------------+
|label|features                                                                  |
+-----+--------------------------------------------------------------------------+
|1.0  |[53.0,0.0,2.0,0.0,2.3396227029999994,13.0,0.0,1.0,0.0,2.5384615384615388]|
|1.0  |[17.0,0.0,28.0,1.0,3.2352942800000006,10.0,0.0,1.0,0.0,3.2]               |
|1.0  |[17.0,0.0,1.0,0.0,3.2352942800000006,11.0,0.0,1.0,0.0,3.0909090909090904]|
|1.0  |[21.0,1.0,2.0,1.0,4.76190486,13.0,1.0,1.0,0.0,2.857142857142857]          |
|1.0  |[5.0,0.0,6.0,0.0,3.4000000000000004,2.0,0.0,1.0,0.0,4.0]                  |
|1.0  |[24.0,0.0,1.0,0.0,2.5416666919999997,3.0,0.0,1.0,0.0,2.666666666666665]   |
|1.0  |[53.0,0.0,13.0,0.0,2.3396227029999994,13.0,0.0,1.0,0.0,2.307692307692308]|
|1.0  |[5.0,0.0,105.0,1.0,2.8,2.0,0.0,1.0,0.0,2.5]                               |
|1.0  |[17.0,0.0,3.0,2.0,3.2352942800000006,12.0,0.0,1.0,0.0,3.1666666666666665]|
```

特征可以划分为批式特征和流式特征。需要注意的是，在模型训练阶段，流式特征从历史数据（HBase）模拟得到（例如以某条记录的前 1000 条记录中提取该记录的流式特征），而在模型预测阶段，从 Redis 中获取流式处理得到的流式特征。

训练结束后，可将模型的参数（lrModel.coefficients、lrModel.intercept）取出并保存在 Redis 中，以供服务端使用。

（6）使用 Spark Streaming 或 Strom 实现流式计算任务

不同于批式计算,流式计算直接从 Kafka 中消费“movie_rating_records” Topic，定时时间为 30 秒(30*1000ms)并完成以下两部分内容的计算:

①实时特征的计算

根据一定时间窗口从 Kafka 中读取实时数据，并计算以下五个流式特征，具体如下表所示。

streaming2feature_userId_rating1_{userId}	用户最近正反馈次数
streaming2feature_userId_rating0_{userId}	用户最近负反馈次数
streaming2feature_userId_to_genresId_{userId}_{genresId}	用户最近点击该分类比例
streaming2feature_movieId_rating1_{movieId}	电影最近被正反馈次数
streaming2feature_movieId_rating0_{movieId}	电影最近被负反馈次数

②实时 topK 的计算

根据一定时间窗口从 Kafka 中读取实时数据,计算出现次数 topK 的 movieId 和每个电影类别中出现 topK 的 movieId，并存入 Redis 中。

popular_movies_all	所有电影 topK
popular_movies_genreId_{genreId}	某类别电影 topK

（7）实现服务应答

服务端根据客户端的请求，从 Redis 中读取多种召回方式得到的候选列表、批式特征、流式特征、CTR 模型参数，根据 CTR 模型参数与特征，为列表中每个电影打分，并将评分最高的 topK 部电影返回给客户端。

（8）（选做内容 1）实现简易后台监控（10 分）

一个系统通常还需要有后台监控与分析能力。可以从以下两方面考虑:

①使用 Hive/SparkSQL 实现简易后台分析统计（5 分）

利用 Hive/SparkSQL，定期统计:不同类别的电影的记录数

②将后台分析统计结果可视化（5 分）

将①的结果可视化,使用条形图、桑基图等进行可视化展示,并定期更新图表。

（9）（选做内容 2）提升推荐系统的推荐效果（10 分）

①增加更多的召回（5 分）

使用多种方式进行召回，如使用实验（四）中的协同过滤模型，对于每个 userId 产生单独的召回列表。

②使用更多的特征（5 分）

进一步从数据中进行挖掘，例如使用上电影的年份等信息。

（10） Redis 的改进（10 分）

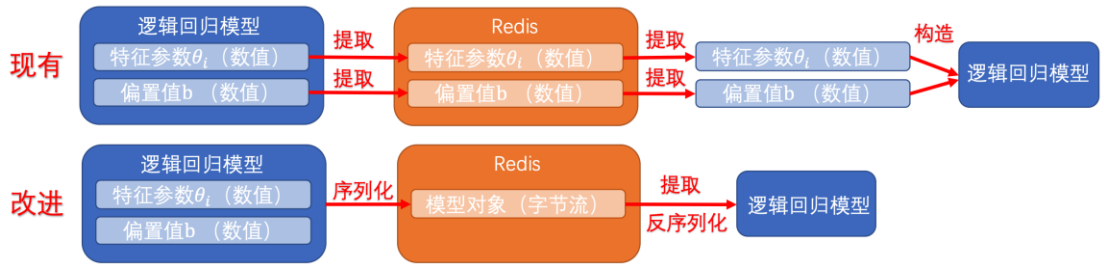
①分布式 Redis 的部署（3 分）

Redis 支持集群化部署，通过增加 Redis 集群的节点可以增加对 Redis 的读取访问速度。本改进要求启动一个至少两个 Redis 进程的集群。

②对于存储在 Redis 中的特征和数据，设置一个合理的过期时间并说明自己的思考（3 分）

③CTR 模型序列化后直接存储 Redis 中（4 分）

本次实验中使用 LR 模型，由于 Redis 不支持对象存储，因为想要在 Redis 中存储该模型，可以通过将模型参数θ_i、b（都为数值型）从模型中提取出来，再分别存储到 Redis 中，然后服务端从 Redis 中提取这些模型参数，并根据这些模型参数重新构造模型用于预测。但对于拥有较多模型参数的算法模型来说，这种做法过于繁琐。更一般的做法是，基于 Redis 支持二进制字节流存储的特性，将整个模型对象序列化后直接存到 Redis 中，在服务端使用时再从 Redis 中提取并反序列化模型对象，即



附加说明：

（1） 数据集说明

数据集采用实验（四）中的“movies”数据集，并经过一定的处理，包含以下三个文件：movies.csv, json_train_ratings.json, json_test_ratings.json。

movies.csv：存储电影信息，包括电影 ID，电影 title，电影 genres，电影 year，以及属于各个类别。在 movies.csv 中，一共出现了 19 个类别的电影，具体见数据文件。在系统运行前，通过提供的 load_movie_redis.py 组件将这部分数据导入到 Redis 中。

json_train_ratings.json：用于模拟历史数据。数据格式为附加说明（2）中的 json 格式。在系统运行前，通过提供的 load_train_rating_hbase.py 组件将这部分数据导入到 HBase 中。

json_test_ratings.json：用于模拟实时数据。数据格式为附加说明（2）中的 json 格式。在系统运行时，通过提供的 generatorRecord.py 组件发送到 Flume。

（2） 数据发生器中的数据格式

“headers”字段：包括一个唯一的字段“key”，值为该记录的主键。

“body”字段：包括一个**字符串**。字符串的内容为 json 格式的字典（对应 java 中的 Map，python 中的 dict），该字典包括四个字段：userId，movieId，rating，timestamp。

```
{
"headers":
    {"key": "1230673202rating"},
"body":
    "{\"userId\": 51, \"movieId\": 59615, \"rating\": 1.0, \"timestamp\": 1230673202}"
}
```

(3) Kafka 中的数据格式

Kafka 中的数据格式类似实验 2，当 Flume 传递给 Kafka 时，会自动将“headers”中的“key”字段作为 Kafka 消息的 key 字段，将“body”作为 Kafka 消息中的 value 字段。

(4) HBase 中的数据格式

HBase 中创建一张表 movie_records，该表包含一个列族“details”，该列族包含 4 个列，“userId”，“movieId”，“rating”，“timestamp”。

```
ROW                          COLUMN+CELL
 1000018415rating            column=details:movieId, timestamp=2021-05-18T22:20:24.716, value=3030
 1000018415rating            column=details:rating, timestamp=2021-05-18T22:20:24.716, value=1.0
 1000018415rating            column=details:timestamp, timestamp=2021-05-18T22:20:24.716, value=1000018415
 1000018415rating            column=details:userId, timestamp=2021-05-18T22:20:24.716, value=448
 1000085668rating            column=details:movieId, timestamp=2021-05-18T22:20:24.718, value=1097
 1000085668rating            column=details:rating, timestamp=2021-05-18T22:20:24.718, value=1.0
 1000085668rating            column=details:timestamp, timestamp=2021-05-18T22:20:24.718, value=1000085668
 1000085668rating            column=details:userId, timestamp=2021-05-18T22:20:24.718, value=45
 1000148915rating            column=details:movieId, timestamp=2021-05-18T22:20:24.719, value=1089
 1000148915rating            column=details:rating, timestamp=2021-05-18T22:20:24.719, value=1.0
 1000148915rating            column=details:timestamp, timestamp=2021-05-18T22:20:24.719, value=1000148915
 1000148915rating            column=details:userId, timestamp=2021-05-18T22:20:24.719, value=452
 1000197419rating            column=details:movieId, timestamp=2021-05-18T22:20:24.720, value=3868
 1000197419rating            column=details:rating, timestamp=2021-05-18T22:20:24.720, value=1.0
 1000197419rating            column=details:timestamp, timestamp=2021-05-18T22:20:24.720, value=1000197419
```

(5) Redis 中的数据格式

Key	Value	说明	作用
popular_movies_all	List	所有电影 topK	召回
popular_movies_genreId_{genreId}	List	某类别电影 topK	召回
*userId_perfer_movieId_{userId}	List	经过协同过滤模型 topK	召回
movie2genres_movieId_{movieId}	List	每个商品对应的类别	信息
streaming2feature_userId_rating1_{userId}	String	用户最近正反馈次数	实时特征
streaming2feature_userId_rating0_{userId}	String	用户最近负反馈次数	实时特征
streaming2feature_userId_to_genresId_{userId}_{genresId}	String	用户最近点击该分类比例	实时特征
streaming2feature_movieId_rating1_{movieId}	String	电影最近被正反馈次数	实时特征
streaming2feature_movieId_rating0_{movieId}	String	电影最近被负反馈次数	实时特征
batch2feature_userId_rating1_{userId}	String	用户历史正反馈次数	批式特征
batch2feature_userId_rating0_{userId}	String	用户历史负反馈次数	批式特征
batch2feature_userId_to_genresId_{userId}_{genresId}	String	用户历史点击该分类比例	批式特征
batch2feature_movieId_rating1_{movieId}	String	电影历史被正反馈次数	批式特征
batch2feature_movieId_rating0_{movieId}	String	电影历史被负反馈次数	批式特征
movieId2movieTitle_{movieId}	String	从电影名映射到电影标题	服务

For example:

```
127.0.0.1:6379> lrange popular_movies_all 0 -1
 1) "6377"
 2) "27803"
 3) "551"
 4) "80831"
 5) "3037"
 6) "296"
 7) "99114"
 8) "1298"
 9) "918"
10) "370"
11) "4816"
12) "858"
13) "59387"
14) "2067"
15) "5989"
16) "8580"
17) "1394"
18) "3114"
19) "318"
20) "858"
21) "38038"
22) "61348"
23) "2959"
24) "2739"
25) "7669"
26) "296"
27) "59784"
28) "364"
29) "589"
30) "1201"
```

```
127.0.0.1:6379> lrange popular_movies_genreId_7 0 -1
1) "1201"
2) "364"
```

```
127.0.0.1:6379> lrange movie2genres_movieId_48596 0 -1
1) "3"
2) "8"
3) "9"
```

```
127.0.0.1:6379> get streaming2feature_userId_rating1_51
"3"
```

```
127.0.0.1:6379> get "batch2feature_userId_to_genresId_13_14"
"0.16666667"
```

```
127.0.0.1:6379> get "movieId2movieTitle_66097"
"Coraline (2009)"
```

注意事项：

（1） 本次实验是小组的形式，一个小组不多于 5 个人。

（2） 本次实验报告提交截止时间为第 17 周，即 6 月 27 号前（包括 6 月 27 号）。

（3） 本次实验最终的分数由以下几个部分组成：

- 架构完整性（70 分）
- 选做内容完成度（10 分，这部分最多 10 分，多做不加分）：
- 实验报告书写（20 分）

（4） 本次实验严禁抄袭。

（5） 本次实验的邮件名与实验报告的命名格式都为“姓名-学号-综合实验报告”。例如，

邮件名：**赵四-2018222333-综合实验报告**

邮件正文：（空）

邮件附件：**赵四-2018222333-综合实验报告.doc**

注意命名格式中各字段的相对顺序与使用的分隔符。除此之外，**不要出现多余的字眼，**如“大数据基础实验”，多余的双引号，多余的班级信息等。不按照要求提交将会扣除

本次实验的分数。

（6） 本次实验要求以附录的形式，将代码文本粘贴在实验报告最后的附录部分。在实验报告的正文部分可以对代码片段进行解释，但不要将所有代码粘贴在正文，也不要以图像的形式粘贴代码。保护多个代码文件时，按照组件的不同对代码进行分类，并从大写字母“A”开始用不同的编号划分开，如实验若包含 kafka、Redis、MySQL 代码，则在附录“A”部分粘贴 kafka 代码文本，在附录“B”部分粘贴 Redis 代码文本，在附录“C”部分粘贴 MySQL 代码文本。