

程序设计竞赛组队推荐系统研究与设计

摘 要

大学生程序设计竞赛（如 ICPC、CCPC 等）在当下的高校中有着非常高的热度，能够有效的锻炼参赛选手的编程能力。竞赛以三人组队形式参加，然而目前学校内的队伍组成还是以相熟的同学互相组队为主，缺乏了组队的多样性和团队知识覆盖的全面性。与此同时，高昂的服务器价格与后续的维护费用阻碍了学校为集训队搭建专门的服务平台。

针对上述问题，本文设计了基于推荐系统和大数据处理技术的程序设计竞赛团队训练系统，为参赛选手提供了组队推荐、题目推荐、个人数据可视化等服务；为参赛团队提供了训练计划列表，团队知识覆盖程度可视化等服务；为学校提供了参赛队伍导出、构建竞赛代码库等服务。

系统主要包括四个部分：数据收集模块、大数据特征处理模块、推荐模块与系统业务模块。对各个模块所用技术以及功能作用说明如下：

（1）数据收集模块：分为离线数据收集与在线数据收集。离线收集采用 go-colly 框架分布式爬取用户在 Codeforces 等程序竞赛网站的提交记录；在线收集使用 zap 与 lumberjack 进行日志的收集与切割。

（2）大数据特征处理模块：离线收集的数据上传到 HDFS 中进行存储。Spark 使用这些数据生成 Embedding 向量供接下来的模型训练使用。

（3）推荐模块：使用 TensorFlow 训练 NerualCF 模型与双塔模型，并通过 TensorFlow Serving 提供 gRPC 和 RESTful API 调用。

（4）系统业务模块：前端使用 Vue.js + Vite + Element-Plus UI 完成业务页面的开发与设计；使用 Go 语言的 Gin 框架完成后端开发，它是连接起整个的系统的桥梁，完成了响应前端请求、记录日志、特征拼装、请求模型服务 API 等任务。

关键词 程序设计竞赛；推荐系统；大数据处理；Embedding；

Research and Design of Team Recommendation System for Programming Competition

Abstract

Collegiate Programming Contest (such as ICPC, CCPC, etc.) has a very high heat, can effectively exercise programming capabilities contestants. The competition is in the form of a three-person team, but currently it is mainly a team of acquaintances, which lacks the diversity of the team and the comprehensiveness of the team's knowledge coverage.

To solve the above problems, this paper designed a team training system based on the recommendation system and big data processing technology to provide team recommendation, title recommendation, personal data visualization and other services for the competitors. Provides the team training program list, team knowledge visualization services covering degree; Provides for the school teams export, build competition code library services.

The system mainly includes four parts: data collection module, big data feature processing module, recommendation module and system business module.

1. Data collection module: Divided into offline data collection and online data collection. Offline collection uses the go-colly framework to crawl user submission records on Codeforces and other program competition websites; real-time collection uses zap and lumberjack to collect and cut logs.

2. Big data feature processing module: Data collected offline is uploaded to Hadoop HDFS storage for Spark's next data processing.

3. Recommended module: Use TensorFlow to train the NerualCF model and the twin tower model, and provide gRPC and RESTful API calls through TensorFlow Serving.

4. System business module: The front-end uses Vue.js + Vite + Element-plus UI to complete the development of business pages; Use Gin framework development after the completion of the end, it completed response front-end request, logging, characteristics, model service API and other tasks.

Keywords Programming contest, Recommendation system, Big data processing, Embedding;

目 录

摘要	I
Abstract	II
第 1 章 绪论	1
1.1 研究背景及意义	1
1.2 国内外研究与发展现状	2
1.2.1 推荐系统研究与发展现状	2
1.2.2 推荐算法在算法竞赛领域的研究现状	4
1.3 论文的主要工作	5
1.4 论文的组织结构	5
第 2 章 相关技术与理论基础	6
2.1 推荐算法理论研究	6
2.1.1 基于内容的推荐算法	6
2.1.2 常用向量相似度计算方法	7
2.1.3 基于协同过滤的推荐算法	7
2.1.4 协同过滤的优化—矩阵分解	9
2.1.5 Embedding 在推荐系统中的应用	11
2.1.6 深度学习在推荐系统中的应用	15
2.2 推荐系统评测	16
2.2.1 评测方法	16
2.2.2 评测指标	16
2.3 本章小结	17
第 3 章 程序设计竞赛组队推荐系统分析	18
3.1 系统需求分析	18
3.1.1 功能性需求	18
3.1.2 非功能性需求	19
3.2 开发环境分析	20
3.2.1 硬件环境	20
3.2.2 软件环境	20
3.3 本章小结	20
第 4 章 程序设计竞赛组队推荐系统设计	21
4.1 系统总体设计	21
4.2 数据收集模块	22
4.2.1 编程语言 Golang	22

4.2.2 爬虫框架 Go-Colly.....	22
4.2.3 定时任务 Cron.....	23
4.2.4 日志收集 Zap.....	23
4.3 大数据特征处理模块.....	24
4.3.1 分布式文件系统 HDFS.....	25
4.3.2 分布式计算平台 Spark.....	25
4.4 推荐模块.....	27
4.4.1 模型训练 TensorFlow.....	27
4.4.2 模型线上服务 TensorFlow Serving	27
4.5 系统业务模块.....	27
4.5.1 前端框架 Vue	27
4.5.2 后端框架 Gin.....	27
4.6 本章小结.....	27
第 5 章 程序设计竞赛组队推荐系统实现.....	28
5.1 环境部署	28
5.1.1 Hadoop 集群部署	28
5.1.2 Spark 集群部署.....	28
5.1.3 Flink 集群部署	28
5.2 算法实现与结果分析.....	28
5.3 系统实现.....	28
5.3.1 数据收集模块实现.....	28
5.4 本章小结.....	28
结论.....	29
致谢.....	30
参考文献.....	31
附录 A.....	33
附录 B	34
附录 C	35

第1章 绪论

1.1 研究背景及意义

据最新统计,我国互联网使用人数达到 9.89 亿,覆盖了全国 70.4%的人口,是世界上最大的互联网使用群体(截止至 2020.12)^[1]。庞大的使用人数,使得如今的互联网有着“人数多、范围广、场景杂”的特点,从而对高校所培养的计算机人才素质提出了更高的要求。

程序千变万化,不变的是它的内在逻辑与思维方式。算法作为程序设计的灵魂所在,在近些年越来越受到国内外高校的重视与青睐。于是各项程序设计竞赛应运而生,高校希望能够通过算法竞赛来以赛促学,激发学生们学习算法的积极性,为学生之后的工作或学术研究打下良好的基础。

国际大学生程序设计竞赛(ICPC)是一项久负盛名的世界性编程赛事,在高校中有着非常高的热度^[2]。参加算法竞赛能够有效的锻炼参赛选手分析问题的建模能力与算法实现的编程能力。竞赛以三人团队形式进行,挑战在 5 小时内编程解决 8-13 道复杂问题,十分考验团队成员的协同配合能力。然而笔者在经过实际的竞赛训练经历与观察之后,发现在训练与组队的环节还存在着若干问题:

(1) 个人训练问题:“题量过大,信息过载”。在程序竞赛高度火热的今天,早已不是 10 年前只有少数几个经典在线测评平台(Online Judge, OJ)的时代了。Codeforces、POJ、HDU 等 OJ 上的题目集都在 7000-8000 道左右,这个数量还在不断增长当中。参赛选手很难在如此庞大的题目集中合理挑选出自己所需要的训练题目。

(2) 团队组合问题:“熟人互组,依赖经验”。在每届新生入学之后,因为互相间还不熟悉,人们更倾向于选择自己的同班同学组队。而对于不同班级之间,缺少一个平台能够让其互相了解编程水平和擅长方向。

(3) 团队训练问题:“缺乏计划,定位不明”。团队在进行训练时,大多都是在被动接受训练安排,缺乏明确且有针对性的训练计划来补齐短板、攻克难关。团队成员在队内的分工方法也时常不明确。

(4) 学校集训队问题:“工作繁琐,资源分散”。教练的工作繁重,从训练计划到比赛安排,经常使用 Excel 表格处理大量数据,数据可视化。整个集训队训练资源分散在 QQ 群中,每一届的队员都会自己制作一些代码模板或经验总结,但是并没有得到很好的应用。

在处理架构上,高昂的服务器价格与后续的维护费用阻碍了学校为集训队搭建专门的服务平台。分布式大数据平台的出现为其提供了一种较为

经济实惠的解决方案。开源框架 Apache Hadoop 可以使用多台廉价的机器组成一个分布式集群，它的三个核心组件 HDFS、Yarn、MapReduce 构成了大数据处理的三驾马车。HDFS 提供了分布式文件存储，Yarn 管理整个集群的资源和任务调度，MapReduce 实现分布式的并行计算。大数据批处理平台 Spark 的出现解决 MapReduce 运行效率低下的痛点，为更快的分布式处理数据提供了支持。这些技术的出现与成熟，为本论文的设计与实现提供了有力支撑。

综上所述，在程序设计竞赛盛行背景下，针对竞赛团队组成过程中所存在的问题，设计基于推荐系统和大数据处理技术的组队推荐系统，对参赛选手们的学习与比赛，有如下几点意义。

(1) 对参赛选手：提供题目推荐，使其不再陷入茫茫题海；提供队友推荐，扩大视野范围，选择更适合自己的队友；个人数据可视化，让长处短板一目了然，更清晰的认知自己的实力。

(2) 对参赛团队：提供了训练计划列表，让团队的实力提升有条不紊；提供团队知识覆盖程度可视化，及时发现存在的短板，取得更好的成绩。

(3) 对集训教练：提供了各个队伍训练情况可视化面板，训练情况轻松掌握；构建集训队知识库，让每一届学生的知识汇聚于此，薪火相传。

(4) 对系统搭建：利用大数据技术，让平台不在依赖昂贵的专用服务器。通过集群也可以支撑起海量日志的存储、处理与分析。

1.2 国内外研究与发展现状

1.2.1 推荐系统研究与发展现状

协同过滤作为推荐算法中影响力最大、应用最广泛的模型，其研究可以追溯到 1992 年^[3]。Xerox 的研究中心开发了一种邮件筛选系统，用来过滤一些用户不感兴趣的无用邮件，协同过滤（Collaborative Filtering, CF）算法第一次出现在了人们的视野中。但是协同过滤在互联网领域大放异彩，还是源于电商巨头 Amazon 对于协同过滤的应用。2003 年，Amazon 发表了工业界的知名论文 Amazon.com Recommenders Item-to-Item Collaborative Filtering^[4]，这不仅让 Amazon 的推荐系统广为人知，更让协同过滤成为很长时间的热点和业界主流的推荐模型。协同过滤可以根据相似度计算对象的不同分为 UserCF 和 ItemCF。其中 UserCF 更符合人们直觉上“兴趣相似的朋友喜欢的物品，我也喜欢”的思想，但是从技术的角度，也存在一些缺点。首先是为了快速找出 Top n 相似用户，维护的用户相似度矩阵会以 n^2 的速度快速增长，对存储系统造成了十分巨大的压力。其次针对用

户稀疏历史行为时的准确率非常低，在用户反馈困难的领域或是新加入的用户效果不佳。而 ItemCF 同样也存在着自己的问题：不具备很强的泛化能力，无法将两个物品相似这一信息推广到其他物品的相似度计算上，从而导致热门物品强烈的头部效应和冷门物品无法得到足够的推荐。

2006 年，矩阵分解算法（Matrix Factorization, MF）在 Netflix 举办的奖金挑战赛中表现出色。主要优点在于其利用矩阵分解解决了 CF 模型在处理稀疏矩阵上的不足。矩阵分解方法分为利用特征值分解、奇异值分解和梯度下降。但 MF 算法依旧没有解决如何将更多的特征（如上下文信息）加入推荐模型中。

逻辑回归模型解决了特征融合的问题，它不再依靠相似度计算去对推荐列表进行排序，而是期望将用户的点击率（Click Through Rate, CTR）概率作为推荐排序的基础。2012 年，阿里巴巴提出将推荐样本进行分片，然后在每片中进行 LR 的运算，最终将加权平均概率作为最后的预估值，该模型被称作大规模分段线性模型（LS-PLM）^[5]，这已经很接近于三层的神经网络，在深度学习的浪潮席卷之前发挥出惊人的效果。但是基于 LR 的推荐模型没有实现交叉特征的处理，有可能在某些情况导致辛普森悖论，出现每种特征的 CTR 概率都很高，但是综合之后反而很低的情况。

为了解决特征交叉的问题，人们从一开始的手动组合到 POLY2 模型将所有特征两两暴力组合，直到 2010 年 FM 模型提出通过引用两个特征隐向量内积作为交叉特征的权重，这才真正解决了特征交叉无法实际应用的问题^[6]。传统推荐模型三大模型的演化关系如图 1-1 所示：

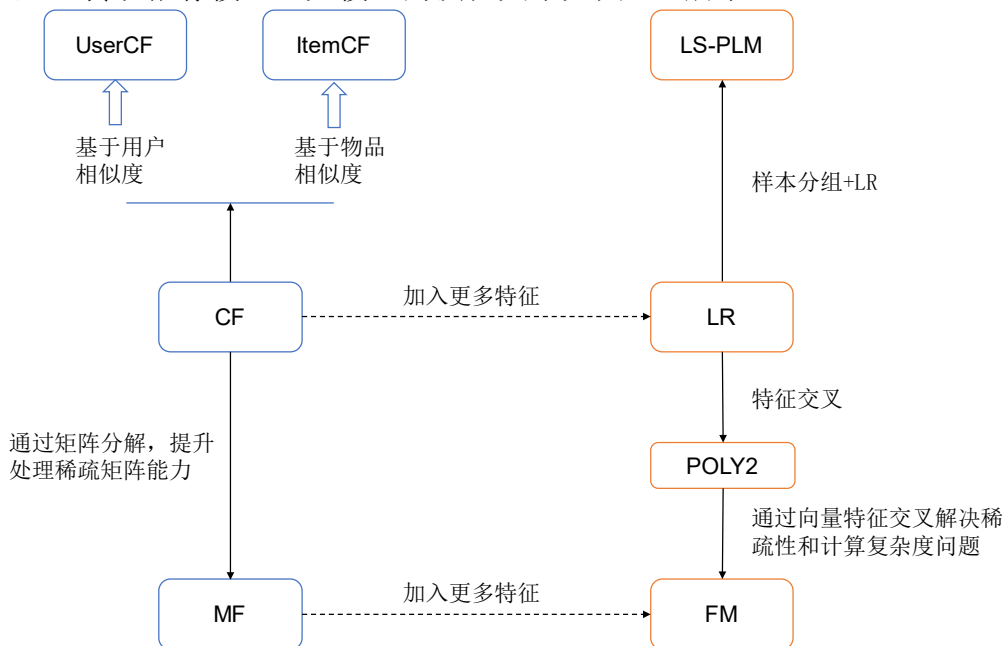


图 1-1 传统推荐模型演化关系图

随着 Alex Krizhevsky 在 2012 年提出了 AlexNet, 开启了深度学习的时代^[7], 推荐算法也深受其影响。而 Google 于 2013 年提出的 Embedding 技术—Word2Vec, 解决了传统特征输入中采用 One-hot 编码导致的矩阵稀疏问题^[8]。Barkan 等人在 2016 年提出的 Item2Vec, 让 Embedding 技术从词扩展到序列, 更加适应了推荐系统的应用场景^[9]。但是这些 Embedding 方法还都是仅仅支持序列信息, 无法覆盖图这种复杂的结构。2014 年 DeepWalk 算法采用随机游走的方法将图模型转化为序列模型, 成功的解决了 Embedding 无法运用在图结构的缺点。但随机游走产生的抽样性不强。这个问题再 2016 年被来自斯坦福大学的 Grover 等人解决^[10], 提出了 Node2Vec 方法, 通过调整游走过程的倾向性来实现相应场景的适应。2018 年阿里巴巴公布其 Graph Embedding 的模型 EGES, 从工程学的视角解决了多 Embedding 的融合问题^[11]。

在推荐算法的实际应用中, 张廉月完成了基于 Flink 流处理框架的电影推荐系统的设计, 其通过分布式大数据计算平台 Apache Spark、Apache Flink 以及多种开源软件, 如文档数据库 MongoDB、缓存服务器 Redis、搜索引擎 Elastic Search、消息中间件 Kafka 等, 搭建完整的大数据处理系统^[12]。景坤使用了推荐系统完成了网约车组队的设计与实现^[13], 胡开冉^[14]提出了基于 FunkSVD 的反向推荐可以大大提高物品的覆盖率, 解决推荐算法中的长尾效应。李斌提出一种基于迁移学习的领域自适应推荐方法用于赛事推荐^[15]。

1.2.2 推荐算法在算法竞赛领域的研究现状

到目前为止, 很少有专注于算法竞赛推荐领域的研究: 2014 年 Toledo 等人提出了一种基于传统协同过滤的方法, 并采用了适合案例的新相似度度量^[16]。Year 等人在 2018 年提出基于模糊逻辑的方法^[17]。Caro 和 Jimenez 等人考虑了基于用户相似性的方法来进行题目的推建^[18]。2020 年 fantozzi 等人使用了欧洲学生在意大利信息学奥林匹克竞赛上的数据训练了基于自动编码器神经网络的 ANN 模型, 并测试两种不同的方法, 离线数据和经过改动的增量数据, 在运用深度学习推荐题目中做出了尝试^[19]。

国内对于在 OJ 中进行题目推荐的尝试有: 2015 年孙权提出使用协同过滤算法进行题目推荐^[20], 2018 年朱国进等人采用的基于关联规则挖掘的方法进行推荐^[21], 2019 年肖春芸等人在华东师范的 EOJ 平台上使用基于用户的协同过滤算法, 来为学生生成推荐题目路径^[22]。知名在线测评平台 LeetCode 也提供了热门题目推荐功能。这些尝试都是基于传统的推荐算法, 受到的局限性较大, 往往无法真正的对每一个训练队员进行个性化的服务。

1.3 论文的主要工作

本文首先分析了目前程序设计竞赛各个环节中存在的不足，从个人训练、团队组合到教练员的繁重工作。然后从三个不同的视角给出对应的解决方法。最后使用了大数据技术和基于 Embedding 与深度学习的推荐算法，在程序设计竞赛组队推荐领域做出了一次尝试。以上三点也是本论文的创新所在。希望该系统设计与实现可以有效的帮助到集训队的同学进行更高效的训练以及减轻教练员重复性的劳动。

系统主要包括四个部分：数据收集模块、大数据特征处理模块、推荐模块与系统业务模块。现对各个模块所用技术以及功能作用说明如下：

(1) 数据收集模块：分为离线数据收集与在线数据收集。离线收集采用 go-colly 框架分布式爬取用户在 Codeforces 等程序竞赛网站的提交记录；在线收集使用 zap 与 lumberjack 进行日志的收集与切割。

(2) 大数据特征处理模块：离线收集的数据上传到 Hadoop HDFS 进行存储。Spark 使用这些数据生成 Embedding 供接下来的模型训练使用。

(3) 推荐模块：使用 TensorFlow 训练 NeuralCF 模型与双塔模型，并通过 TensorFlow Serving 提供 gRPC 和 RESTful API 调用。

(4) 系统业务模块：前端使用 Vue.js + Vite + Element-Plus UI 完成业务页面的开发与设计；使用 Go 语言的 Gin 框架完成后端开发，它是连接起整个的系统的桥梁，完成了响应前端请求、记录日志、特征拼装、请求模型服务 API 等任务。

1.4 论文的组织结构

本文共分为 6 章，各章节的安排如下：

第一章为绪论，首先介绍了本论文的研究背景及其意义。然后对推荐系统国内外的研究发展进行了梳理，调查了目前推荐算法在程序设计竞赛领域的研究进展。最后简要说明本文的主要工作和组织结构安排。

第二章为相关技术和理论基础，首先对推荐算法的原理及数学推导做了详细的说明，然后对其评测指标做了简要介绍。主要介绍的算法有基于内容的推荐、协同过滤、矩阵分解、Embedding 技术和 NeuralCF 模型。

第三章为系统分析。在进行系统的详细设计之前，进行需求分析来确定系统最终要实现的功能。最后介绍了系统运行所需的软、硬件资源。

第四章为系统设计。

第五章为系统实现与结果分析。

第2章 相关技术与理论基础

2.1 推荐算法理论研究

2.1.1 基于内容的推荐算法

基于内容的推荐算法结构简单、历史悠久，如今依旧在很多场景下表现良好。整个算法的核心思想就是为用户 X 推荐之前喜欢的物品相似的物品列表 Y。下图 2-1 展示了基于内容推荐算法的 Codeforces 题目推荐流程：

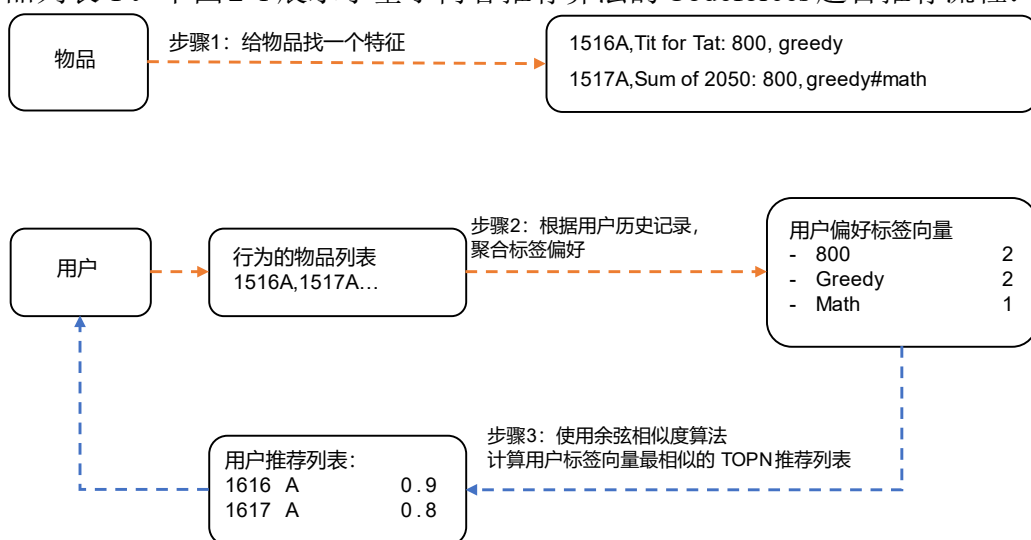


图 2-1 基于内容推荐算法流程

（1）**步骤一**：为每个物品找一个特征，如难度系数、分类标签等。

（2）**步骤二**：收集用户的历史提交记录，比如用户 A 做过 1516A、1517A 这两道题，将题目列表标签做聚合统计，形成用户的偏好标签向量。

（3）**步骤三**：利用余弦相似度算法，计算与用户的偏好标签向量最接近的 TopN 列表（如上图 1616A 这道题和用户的偏好相似程度 90%，非常可能在选择做），然后推荐给用户。

其优缺点分析如下表所示：

表 2-1 基于内容推荐算法优缺点分析

优点	缺点
不需要其他用户的数据	一直处在舒适圈之中，不断推荐水题
不存在冷启动问题	推荐的同质化现象严重
推荐结果直观易于解释	无法与更多的特征交叉融合

2.1.2 常用向量相似度计算方法

在上文步骤三中提到了计算用户偏好向量和物品向量的相似程度，下面列举出几种常用的相似度计算方式，并对其含义进行说明。

(1) **余弦相似度**：如（式 2-1）所示，通过度量两个向量夹角的 $\cos(\theta)$ 值来表示向量之间的相似程度。余弦相似度计算简单迅速，并且在高维的情况下依旧适用。

$$\cos(\theta) = \frac{\sum_{k=1}^n x_{1k}x_{2k}}{\sqrt{\sum_{k=1}^n x_{1k}^2} \sqrt{\sum_{k=1}^n x_{2k}^2}} \quad (2-1)$$

(2) **皮尔逊相关系数**：如（式 2-2）所示，在利用余弦相似度进行计算的时候，若用户没有评价的物品分数会被直接判 0，这么做的后果会导致原本高分和低分的物品在计算完相似度后特别靠近。而皮尔逊相关系数将用户还未评分的物品填充样本均值，然后再计算余弦相似度。

$$\text{sim}(x_1, x_2) = \frac{\sum_{k=1}^n (x_{1k} - \bar{x}_1)(x_{2k} - \bar{x}_2)}{\sqrt{\sum_{k=1}^n (x_{1k} - \bar{x}_1)^2} \sqrt{\sum_{k=1}^n (x_{2k} - \bar{x}_2)^2}} \quad (2-2)$$

(3) **Jaccard 相关系数**：如（式 2-3）所示，杰卡德相似系数从集合的角度衡量相似度，计算方法是看相同元素占总体的比例。而杰卡德距离（式 2-4）与之恰恰相反，用不同元素占总体比例来衡量两个集合的区分度。

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (2-3)$$

$$J_\delta(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \quad (2-4)$$

无论选取哪种相似度计算方式都需要具体问题具体分析，灵活的相似度算法使用可以帮助我们有效提升推荐的效果。

2.1.3 基于协同过滤的推荐算法

协同过滤（Collaborative Filtering）的推荐依据是使用行为数据，利用集体的智慧来进行推荐。UserCF 与 ItemCF 的区别与联系如下图 2-2 所示：UserCF 是根据找到和你兴趣接近的人，将他们喜欢的其他物品推荐给你，ItemCF 是找到与你所喜欢物品类似的物品，然后推荐给你。左图中 User1、User2、User4 都喜欢 Item1 和 Item2，那么认为这三个人的兴趣相同，当 User2 和 User4 新喜欢一个物品 Item4 时，便把此物品推荐给 User1。右图

中 User2 和 4 同时喜欢 Item2 和 Item3，系统便认为 Item2 和 3 是相似物品。当 User1 喜欢 Item2 时，系统也会把 Item3 也推荐给他。

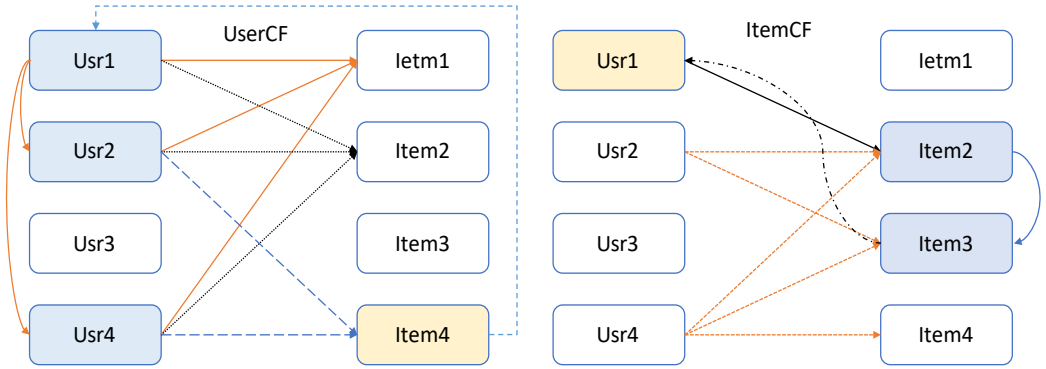


图 2-2 UserCF 与 ItemCF 原理

在获得了 TopN 个相似用户之后，我们就可以轻易的获得用户对于其还没有评价的物品的预估值。其实现方式通常为利用其他用户 s 对该物品 p 的评分的加权评分，权重 $w_{u,s}$ 为其他用户 s 与推荐用户 u 的相似度，最终得到用户 u 对于物品 p 的预估评分 $R_{u,p}$ 。整个过程如（式 2-5）所示，

$$R_{u,p} = \frac{\sum_{s \in S} (w_{u,s} \cdot R_{s,p})}{\sum_{s \in S} w_{u,s}} \quad (2-5)$$

协同过滤算法虽然经典，但是缺点也是显而易见的，比如上图 2-2 里的 Uer3，当他还未对某个物品进行评分时就无法找到与他相似的用户，这就是推荐系统中常见的冷启动问题。还有在物品数量和用户数量特别多的时候，维护用户-物品评分矩阵对存储的压力极大，所以许多公司选择了 ItemCF 作为其推荐算法。

ItemCF 的具体流程如下：

- （1）获得用户的历史行为信息，组成 $M \times N$ 维的用户-物品矩阵
- （2）以列为向量，通过皮尔逊相关系数计算物品之间的相似程度，最终构建出 $N \times N$ 维的物品相似度矩阵。
- （3）获得用户的喜爱列表（正反馈可以靠与平均值相比来界定）
- （4）针对目标用户 u 的喜爱列表，分别每个对已评价物品生成 TopN 个相似物品，组成相似物品候选集。
- （5）对相似物品候选集里面的物品，对其按照加权后的总相似度进行排序，最终生成排序列表。加权公式如（式 2-6）所示：

$$R_{u,p} = \sum_{h \in H} (w_{p,h} \cdot R_{u,h}) \quad (2-6)$$

其中 $R_{u,p}$ 是目标用户 u 对未评分物品 p 的预估情况， H 是用户 u 的喜

爱物品集合，权重 $w_{p,h}$ 是未评分物品 p 与已评分物品 h 的相似度， $R_{u,h}$ 是用户 u 对 h 的评分。

下表 2-2 是 UserCF 和 ItemCF 各自的优缺点与适用场景

表 2-2 UserCF 与 ItemCF 优缺点分析

	优点	缺点
UserCF	1. 基于用户相似度进行推荐，有着更强的社交属性。 2. 适用于发现热点，以及跟踪热点的发展趋势。	1. 用户的历史行为可能非常稀疏，找到相似用户的准确率很低。 2. 用户相似度矩阵的维护非常消耗资源，在线存储系统难承受。
ItemCF	1. 对冷启动优化更好，更适用于兴趣变化较稳定的应用	1. 泛化能力较差，冷门的物品会很少被推荐到

2.1.4 协同过滤的优化—矩阵分解

随着用户和物品的不断增多，由用户-物品所构成的矩阵会以 $O(N^2)$ 的速度不断增大，用户的历史行为数据也随着被稀释。矩阵分解作为协同过滤算法的优化版本，其算法原理如下图 2-3 所示：

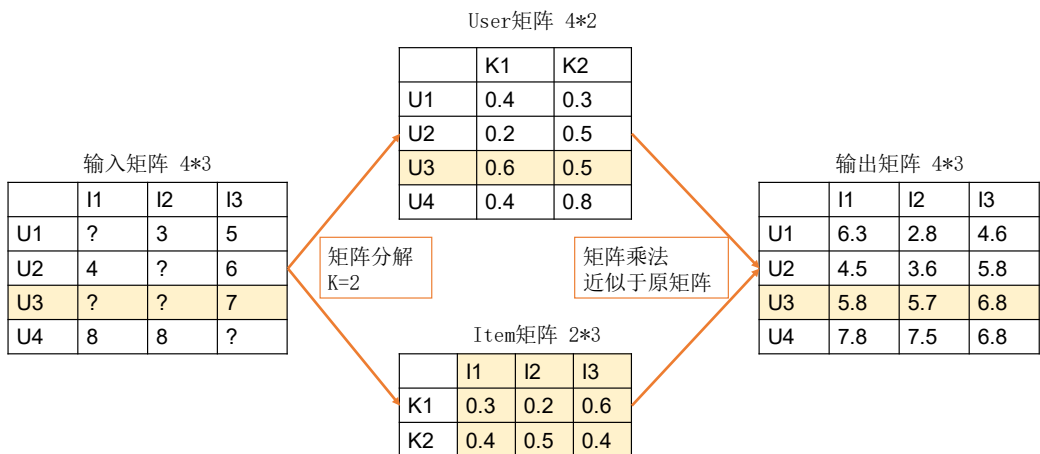


图 2-3 矩阵分解原理

矩阵分解首先将输入矩阵分解成为为两个子矩阵（User 矩阵与 Item 矩阵）。其中若用户数为 M ，物品数为 N ，则 User 矩阵为 $M \times K$ 维，Item 矩阵为 $K \times N$ 维， K 表示特征的数量。分解所得到的 $K1, K2$ 向量被称之为隐向量，它们用来将用户的兴趣和物品特征关联起来。而输出矩阵通过两个分解矩阵相乘后每个单元格都有值，而这些缺失值的填充值就是对用户还未评价物品的预测值。在相乘之后，输出矩阵不一定和原矩阵一致，而我们可以多轮训练，利用梯度下降的方法不断调整 $K1, K2$ 的参数，最终让这个差值降到最小。

矩阵分解模型最终的输出是在分解后最终产生的用户特征向量和物品

特征向量，它们都是 K 个维度，代表着 K 个不同的隐含兴趣点。得到用户和物品的隐向量之后，就可以通过相似度计算来进行排序，最终得到我们所需要的推荐列表。

\hat{r}_{ui} 表示用户 u 对物品 i 的预估评分，若将其表示两矩阵相乘，则如（式 2-7）所示：其中 q_i^T 表示物品 i 在 Item 矩阵中所对应的列向量， p_u 表示用户 u 在 User 矩阵中对应的行向量。

$$\hat{r}_{ui} = q_i^T p_u \quad (2-7)$$

我们求解矩阵分解的目标函数如下（式 2-8）所示：其表示了对于 K 维的用户矩阵 p^* 和物品矩阵 q^* ，让用户已评分和通过隐向量乘积生成的“预估评分”差值最小，将预估的值尽可能的精准。

$$\min_{q^*, p^*} \sum_{(u,i) \in K} (r_{ui} - q_i^T p_u)^2 \quad (2-8)$$

为了避免部分噪声点的影响，减少过拟合现状，对（式 2-8）加入 $L2$ 范数作为正则化项后如（式 2-9）所示： λ 为正则化系数，代表约束模型稳定性的能力。

$$\min_{q^*, p^*} \sum_{(u,i) \in K} (r_{ui} - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2) \quad (2-9)$$

在确定损失函数（式 2-9）之后，可以利用梯度下降求解最优化问题。首先，对 q_i 求偏导的结果如（式 2-10）所示：

$$2(r_{ui} - q_i^T p_u) p_u - 2\lambda q_i \quad (2-10)$$

对 p_u 求偏导的结果如（式 2-11）所示：

$$2(r_{ui} - q_i^T p_u) q_i - 2\lambda p_u \quad (2-11)$$

利用上式的偏导结果，沿着梯度的反方向更新参数，新的参数等于旧的参数减去学习率*变化率： γ 称为学习率，也叫做学习步长，而刚刚求得的偏导便是参数在这个方向的变化率。更新公式如（式 2-12、2-13）所示：

$$q_i \leftarrow q_i - \gamma \left((r_{ui} - q_i^T p_u) p_u - \lambda q_i \right) \quad (2-12)$$

$$p_u \leftarrow p_u - \gamma \left((r_{ui} - q_i^T p_u) q_i - \lambda p_u \right) \quad (2-13)$$

在不断进行梯度下降的过程中，满足两种情况即可终止训练。一是训练的轮数超过了我们预定义的次数；二是损失值（Loss）已经低于一个限度。即可结束训练，否则不断重复上述的梯度反向传播的过程。

下表 2-3 展示了矩阵分解算法的优缺点

表 2-3 矩阵分解的优缺点分析

优点	缺点
将高维的矩阵映射成多个低维矩阵的乘积，解决了数据稀疏性的问题	推荐结果无法直观的进行解释，其隐空间的维度无法与现实中的概念结合起来
最终的预测精度比较高	模型训练比较耗时，通常以天为粒度更新

2.1.5 Embedding 在推荐系统中的应用

(1) Embedding 技术是什么

Embedding 技术是近几年推荐系统研究的前沿，大量的论文都在攻克这一技术方向。最难能可贵的是，其在真实的应用中收获了非常好的效果。

Embedding 从直观上来讲就是一个向量，里面的元素都是小数，如[0.8, 0.9]就是一个二维的 Embedding 向量，若有 100 个小数，就是一个一百维的 Embedding 向量。

(2) Embedding 技术怎么用

从实际意义上来讲，每个维度上的小数表示了一个强度或是概率。如数组第一维表示数据结构、第二位表示图论。那么用户 Embedding[0.8, 0.9]表示这个人擅长 0.8 强度的数据结构，0.9 强度的图论。物品 Embedding[0.4, 0.6]表示这道题目有 0.4 的强度是数据结构、0.6 的强度是图论。通过余弦相似度([0.8, 0.9],[0.4, 0.6])就可以计算出该用户做出这道题目的概率。同理，我们也可以轻松的通过运算得出用户之间的相似程度和物品之间的相似程度。但一般来说 Embedding 的维度都是成百上千维，很难用实际的含义去对应每一维数字代表的含义。虽然无法解释，但是这样的隐含兴趣向量就和 2.1.5 节中进行的矩阵分解时生成的隐向量一致，可以在推荐领域发挥出巨大的价值。

下表 2-4 展示了 Embedding 的应用范围：

表 2-4 Embedding 向量的用途

序号	Embedding 向量的用途
用途一：	物-物推荐，相关推荐、看了又看、做了又做
用途二：	人-物推荐，猜你喜欢、猜你想看、猜你想练
用途三：	人-人推荐，猜你认识、推荐队友、找到同好

(3) 如何生成 Embedding 向量

生成 Embedding 向量的方式主要有三种：第一种是基于内容的 Word2Vec 方法以及后续发展出来的 Item2Vec、Node2Vec 等词嵌入方式；第二种方法为本文在 2.1.4 小节中介绍的矩阵分解，其中间生成的隐向量也是一种 Embedding 向量；最后一种是直接基于 DNN 深度学习的生成方法。

在介绍 Word2Vec 算法之前，需要先说明一下在其出现之前人们实现词向量的方法——One-Hot 编码。

One-Hot 编码将词典集合 V 中的词映射为一个 V 维向量，将其本身设为 1，其他位置设为 0。这样就可以通过编码唯一的表示一个词。“I Love ICPC, I Like Code.” 的 One-Hot 编码如表 2-5 所示：

表 2-5 One-Hot 编码举例

词	One-Hot 编码
I	1 0 0 0
Love	0 1 0 0
ICPC	0 0 1 0
Like	0 0 0 1
Code	0 0 0 1

这种编码的优点就是简单，只需要将每个词 Hash 一下就可以为其分配编号，但是随着词数量的增多，One-Hot 编码的缺点也显而易见。主要的缺点有三点：

1. **维度膨胀极快**：随着语料库的增加，编码长度可能突破上亿维。
2. **向量之间没有联系**：无法通过 One-Hot 编码之间的相似度运算得到任何有用的信息。
3. **数据稀疏**。经过 One-Hot 编码后的向量几乎全都为 0。

以 Word2Vec 为代表的 Embedding 技术的出现解决了上面所描述的问题。基本思想为将低维空间中的 V 维稀疏 One-Hot 向量映射到高维空间中长度为 N 维的一个点。这里 N 是我们指定的一个参数，表示 Embedding 向量的长度。然后通过简单的点积运算就可以得到两个 Embedding 向量的相似度。Word2Vec 有两种经典的模型：CBOW 和 Skip-gram。它们的结构如图 2-4 所示：

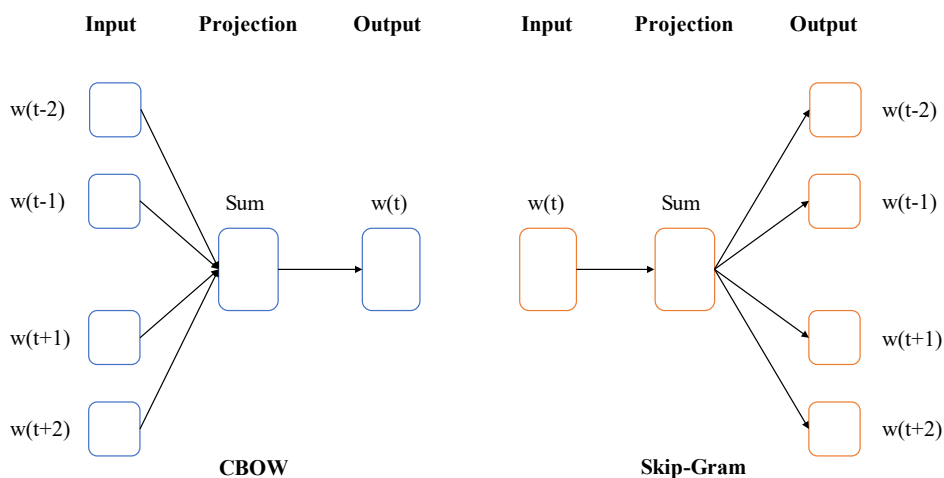


图 2-4 CBOW 模型与 Skip-Gram 模型

这两种模型的区别就在于，CBOW 模型是通过中心词 w_t 周围的词 $w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2}$ 来预测中心词；Skip-gram 模型是通过中心词 w_t 来预测周围词 $w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2}$ 。

CBOW 简化版的模型（仅输入一个词）如下图所示：

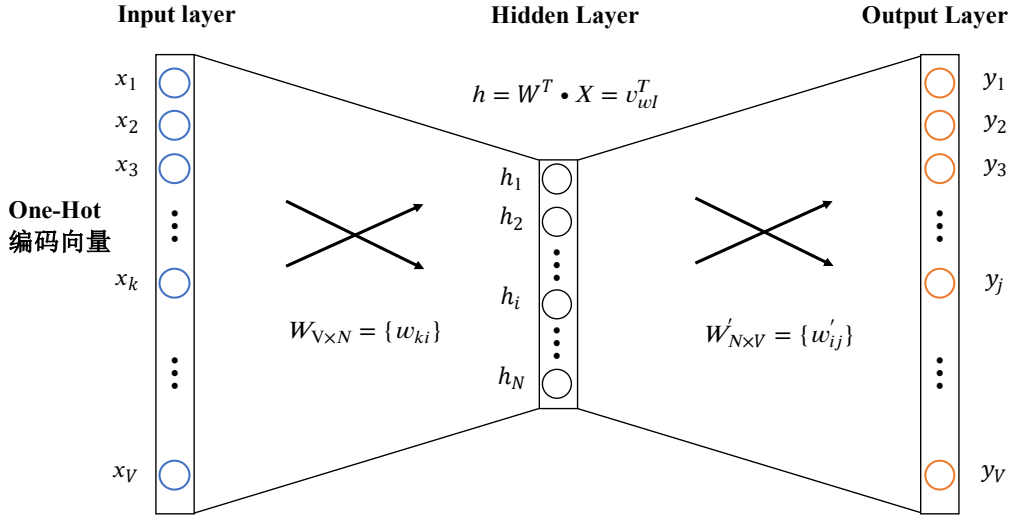


图 2-5 CBOW 简化版模型

1. **输入层：**约定输入的词库大小为 V ，中间的隐层大小为 N 。若将输入单词标记为 w_I ，则输入层是其经过 One-hot 编码后的向量表示，一共有 V 个神经元，除了第 i 个神经元 x_i 为 1，其余各项均为 0，例如 $w_3 = [0, 0, 1, \dots, 0]^T$ 。

2. **权重矩阵 $W_{V \times N}$ ：**输入层向量到隐层之间的权值可以用一个 $V \times N$ 的矩阵 W 表示，其每一行的值对应着一个我们最终需要的 Embedding 向量。将 One-Hot 编码过后的向量 v_w 映射为一个 N 维的短向量 v_{wl}^T ，在这里我们实现了 One-Hot 编码的降维。

3. **隐层：**隐层得到的结果是由输入向量乘权重矩阵得到的。若 One-Hot 编码的第 k 位为 1，相乘后隐层的值等于权重矩阵的第 k 行。其数学形式如（式 2-14）所示：

$$h = W^T \cdot X = W_{k..}^T = v_{wl}^T \quad (2-14)$$

4. **权重矩阵 $W'_{N \times V}$ ：**在隐层到输出层之间有另一个权重矩阵 $W' = \{w'_{ij}\}$ 。通过这个权重矩阵我们可以预估出集合 V 中每个词可能的得分 $u_j, j \in [0, V]$ 。其公式如（式 2-15）所示：

$$u_j = v_{wj}^T \cdot h \quad (2-15)$$

其中 v_{wj}^T 表示权重矩阵 W' 的第 j 列。

5. **输出层：**在输出时将每一个词的得分 u_j 通过 softmax 函数归一化得到一个在 $[0, 1]$ 之间得概率 y_j ，其表示为在输入词为 w_I 时输出词是 w_j 的概率。

可以用（式 2-16）对其进行表示：

$$p(w_j | w_I) = y_j = \frac{\exp(u_j)}{\sum_{k=1}^V \exp(u_k)} \quad (2-16)$$

将（式 2-14、2-15）带入（式 2-16）整理可得（式 2-17）：其中 v'_w 和 v_w 分别是输入词 w 的两种不同的 N 维向量表示形式。

$$p(w_j | w_I) = \frac{\exp(v'_{w_j}{}^T v_{w_I})}{\sum_{k=1}^V \exp(v'_{w_k}{}^T v_{w_I})} \quad (2-17)$$

$$\begin{aligned} L &= \max p(w_O | w_I) \\ &= \max \log y_{j^*} \\ &= u_{j^*} - \log \sum_{k=1}^V \exp(u_k) \end{aligned} \quad (2-18)$$

确定了目标函数后，就可以利用梯度下降和链式求导规则，对两个权重矩阵进行更新，整个下降的过程如图 2-16 所示：

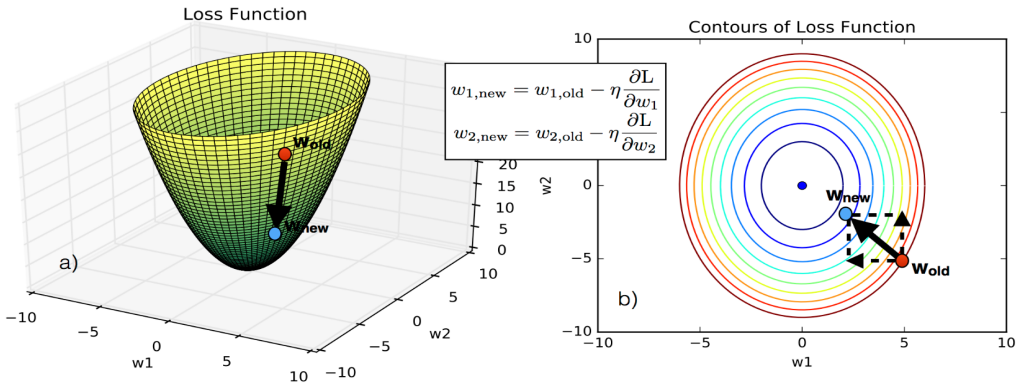


图 2-6 Word2Vec 梯度下降求最优权值

当我们有了 Word2Vec 之后，若将其（文本 ID，文本内容）的输入变为（用户 ID，用户提交 ID）就可以得到一个行为上的 Embedding 向量，这就是 Item2Vec 算法。

（4）如何快速搜索 Embedding 向量

在生成完 Embedding 向量之后，如何快速得到每个用户最相似的 TopN 列表有三种方法。第一种方法就是离线使用两层循环暴力为每个用户的 Embedding 向量挨个计算余弦相似度，然后缓存到 Redis 里面。这种方法在数据量小的时候还可以使用，一旦数据量过大，离线计算的时间就是我们不能接受的。第二种方法是通过离线局部敏感哈希（LSH）来进行相似搜索。其原理是将 Embedding 向量分为很多个桶，然后在桶和桶的附近进

行搜索，可以极大的增加搜索速度。第三种方法为使用 Facebook 开源的近邻搜索库 Faiss，可以实现 ms 级别的在线搜索（百万级别）。

下图 2-7 展示了局部敏感哈希高维空间点向低维空间映射后的结果：

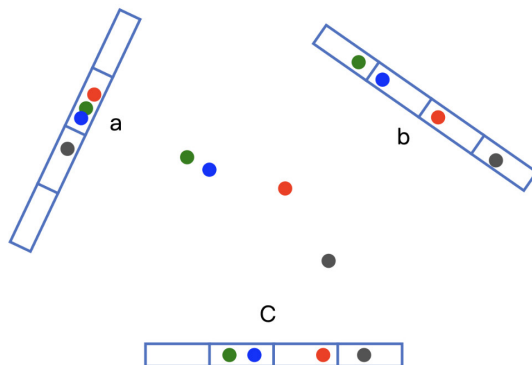


图 2-7 高维空间点向低维空间映射

上文提到了 Embedding 向量是数据经过 One-Hot 编码后映射到高维空间的一个点，而局部敏感哈希就是将其压缩回低维的欧式空间，其原本在高维中相似的点映射到低维依旧靠近。但是原本相隔“距离”较远的点可能会变成接近的点。

2.1.6 深度学习在推荐系统中的应用

如果将矩阵分解算法看做一个简单的神经网络，那么其结构如图 2-8 所示：

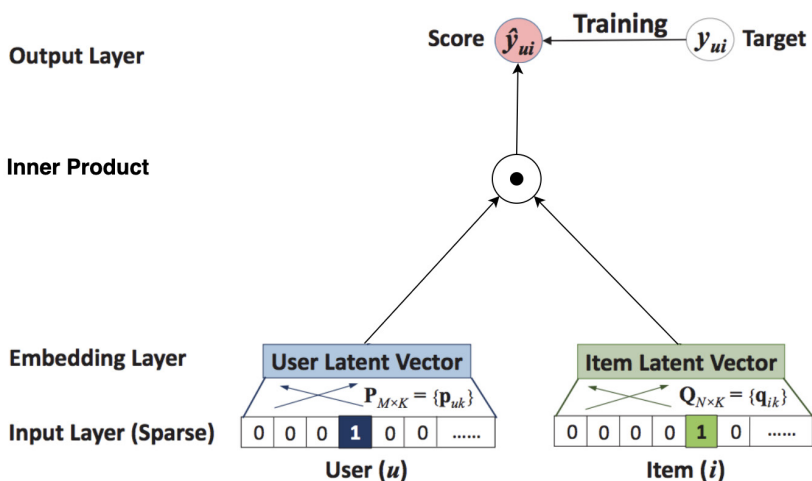


图 2-8 矩阵分解的神经网络结构

输入稀疏的 One-Hot 向量，经过分解得到 Embedding 层，然后通过点积运算得到预测的得分矩阵，最后进行梯度下降来更新 Embedding 层。但是，矩阵分解在 Embedding 层之上直接利用内积得出最终结果，会导致特征之间还没有得到充分交叉，会有欠拟合的风险。

NeuralCF 将简单的点积运算变成了多层的神经网络，提高了模型的拟合能力。其结构如图 2-9 所示：

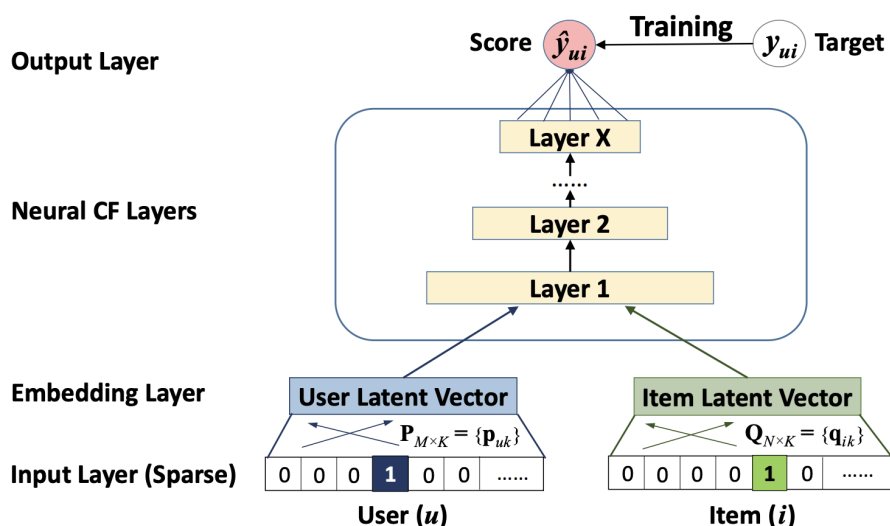


图 2-9 NeuralCF 神经网络结构

NeuralCF 还需要补充

2.2 推荐系统评测

2.2.1 评测方法

2.2.2 评测指标

预测准确度通过比较推荐模型给出的预测评分与用户的真实评分之间的差距，旨在评估推荐模型与用户的真实兴趣偏好之间的拟合程度。自然，差距越小，表示推荐算法的预测准确性越好。常用的预测准确度评价公式有平均绝对误差(Mean Absolute Error, MAE)和均方根误差(Root Mean Squared Error, RMSE)。

有时候，知道用户对哪些物品内容感兴趣比能够预测出用户对物品的评分更加重要，因为预测评分高不代表用户一定会选择（观看、购买）该物品。因此，分类准确度一般用在 Top N 推荐的场景中，并使用准确率和召回率这两个指标进行评估

一般情况下，准确率和召回率呈负相关关系，我们自然希望两个指标都能取得比较高的结果。因此，为了综合评估准确率和召回率，还会使用 F1 指标

2.3 本章小结

首先对推荐算法的原理及数学推导做了详细的说明，然后对其评测指标做了简要介绍。主要介绍的算法有基于内容的推荐、协同过滤、矩阵分解、Embedding 技术和 NeuralCF 模型。

第3章 程序设计竞赛组队推荐系统分析

3.1 系统需求分析

需求分析是在进行系统整体设计之前的一个重要环节，它划清了本系统最终功能实现的范围边界。首先需要对各个模块功能（功能性需求）做出描述，然后对系统在稳定性等非功能性需求上给出约束。

3.1.1 功能性需求

功能性需求是指有具体的完成内容的需求，是一个系统必须具备的能力或者遵从的条件。本程序设计竞赛组队推荐系统的功能如下表 3-1 所示：

表 3-1 功能性需求

系统模块	功能性需求	具体描述
数据收集模块	离线数据收集	爬取用户在 Codeforces 等网站上的历史提交记录，以及整个 Codeforces 上所有的题目数据信息
	在线日志收集	记录用户在系统中的所有操作，如查看文章、提交模板库的修改等操作
分布式处理模块	分布式数据存储	将所有收集的来的数据，分布式的存储在三台服务器构成的 HDFS 系统中，提供给数据处理使用
	分布式数据计算	利用分布式的性能优势，进行数据清洗、One-hot 编码，Embedding 向量生成等操作。
推荐模块	题目个性推荐	针对选手在 Codeforces 网站的历史做题记录，生成符合选手做题风格和难度的推荐列表
	组队队友推荐	计算所有选手之间擅长领域的相似程度，选取中等相似区间的未组队选手进行推荐；若组成 2 人队伍之后以两个人的融合特征做相似计算，为队伍推荐最后一名队员。
	算法新人入门	将所有难度系数 800 分的题目按照做过题目的人数进行排序，生成推荐列表，帮助新人快速熟悉比赛
	算法进阶之路	将各个算法类别下的题目按照难度层级进行分层，并在每层下推荐 10 道做过人数最多的题目。
	近期热门题目	将最近一个月时间所有比赛题目中通过和讨论加权次数最多的 10 道题目生成推荐列表
	历史热门题目	将 codeforces 历史上通过和讨论加权次数最多的 10 道题目生成推荐列表

系统业务 模块	用户注册	本推荐系统需要用户提供电子邮箱和登录密码以进行用户注册，完成注册之后方可登录并使用系统。
	用户登录	用户需要输入正确的电子邮箱、登录密码以及确认密码，完成系统登录。并使用 token 保存用户的登录状态使其不必重复登入
	用户登出	提供用户登出功能
	忘记密码	用户需要通过电子邮箱获取验证邮件，完成密码的找回。
	个人空间	将个人的训练数据以柱形图、饼图、雷达图等可视化方法展现
	团队空间	提供了训练计划列表，让团队的实力提升有条不紊；提供团队知识覆盖程度可视化，及时发现存在的短板，取得更好的成绩。
	集训空间	可视化查看各个队伍的训练情况，如近期通过题目数量、AC 题目的平均分数等。
	团队 Wiki	提供一个 Wiki 平台，允许用户通过提交 markdown 语法的文本进行 Pull Request 请求

3.1.2 非功能性需求

本程序设计组队推荐系统在非功能性需求方面主要有以下几点：

(1) 易用性：整个系统风格一致、操作简单，不需要专业的数据分析能力即可看出自己的擅长方向和不足之处。但是部分功能还是需要一定的计算机专业素养，如提交 Wiki 修改时需要懂得如何通过 Github 提交 PR 请求。

(2) 通用性：整个系统应该能够在 Windows、Linux、Mac 等操作系统上运行。并对访问该系统的电脑无硬件性能要求。

(3) 安全性：在构建系统时，要能够保证用户的数据不会泄露。具体可包括密码的安全性、服务器抓取的离线数据、训练生成的模型文件等不会遭到数据泄露。

(4) 可靠性与可扩展性：保证服务的稳定性，不会出现无法访问等严重事故，保证用户的历史行为记录不会丢失。对项目进行合理的划分，将其分为多个模块，并配合详细的项目文档，以便后续版本的更新和加入更多有志于扩展系统的同学进行多人协同开发。

(5) 成熟性和先进性：在构建整个系统的过程中，使用成熟可靠的软件开发工具，并做到与时俱进，使用业界主流的技术。

3.2 开发环境分析

3.2.1 硬件环境

本系统实现过程中需要采用的硬件配置和其功能如下表 3-2 所示：

表 3-2 硬件配置与功能

名称	配置	作用
Local	Windows 10, 4C16G, RTX 1050Ti-4G	前端、后端开发
Master	CentOS 8.0, 2C4G, 无	Namenode, RM, Master
Servant1	CentOS 8.0, 1C2G, 无	Datanode, Worker
Servant2	CentOS 8.0, 1C2G, 无	Datanode, Worker
GPU Server	Ubuntu 18.04, 16C62G, RTX 3090-24G	训练神经网络

注：配置项中 C 代表 CPU 核数、G 代表内存大小，显卡型号后为显存大小。

3.2.2 软件环境

本系统实现过程中需要采用的软件环境和其功能如下表 3-3 所示：

表 3-3 软件环境及其作用

名称	作用
Python 3.8.5	进行 spark 或 tensorflow 编程
Go 1.16	开发分布式爬虫、构建网站服务端
JDK 1.8	Hadoop 平台运行环境
Scala 1.11	Spark 平台运行环境
Node.js 16.0	基于 Chrome V8 引擎的 JS 解释器
Vue.js 3.0	前端框架
MySQL 8.0	网站业务数据库
Redis 6.2.3	缓存数据库

3.3 本章小结

本章从软件工程的角度出发，对整个系统进行了需求分析，确定了系统最终要实现的功能。最后介绍了系统运行所需的软、硬件资源。

第4章 程序设计竞赛组队推荐系统设计

4.1 系统总体设计

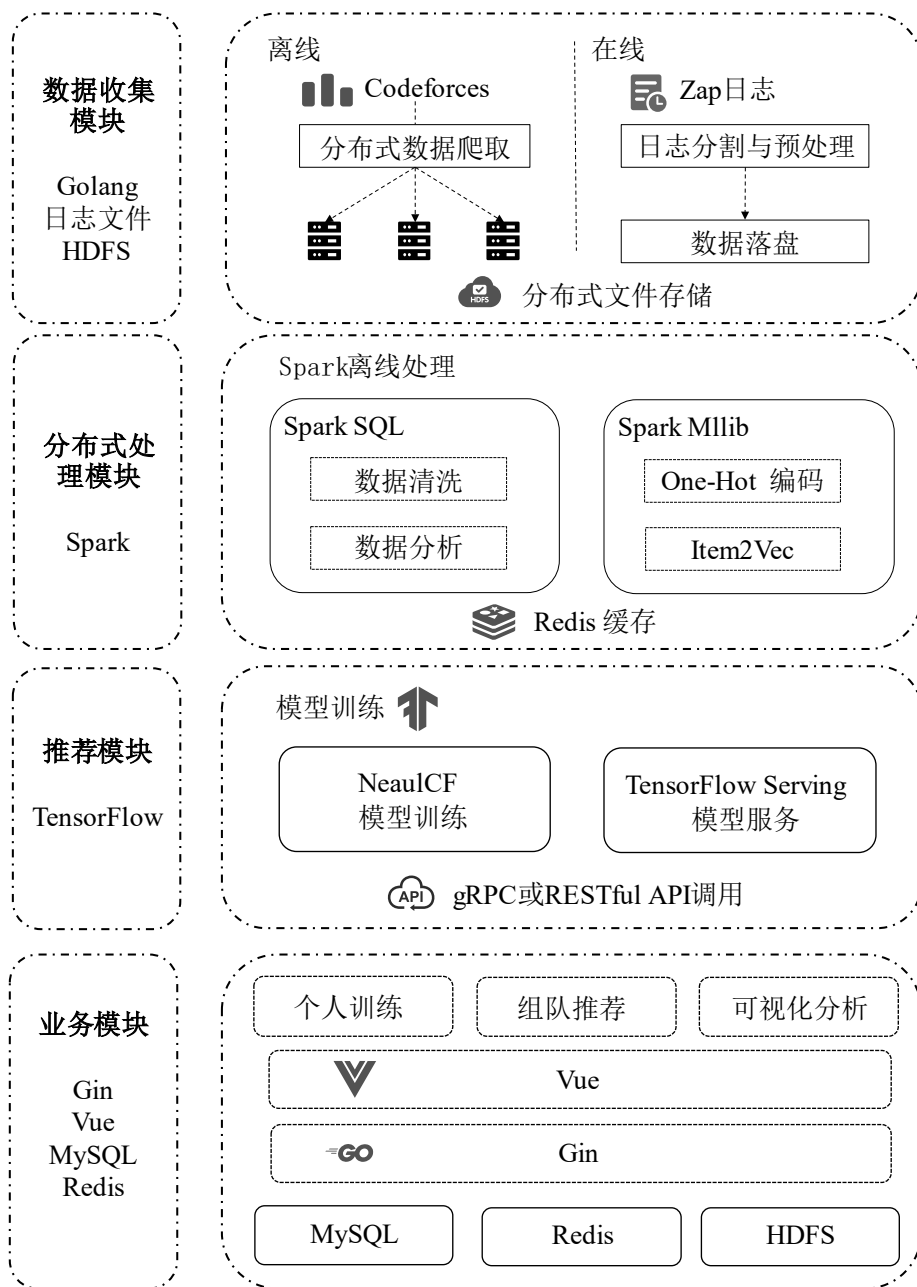


图 4-1 总体技术架构

4.2 数据收集模块

谷歌的研发总监 Peter Norvig 曾说过：“更多的数据优于更好的算法，而好的数据优于多的数据”。这句话强调了数据对于整个系统至关重要。本次设计需要使用离线与在线的数据为队员们进行推荐，离线数据主要来源于 Codeforces 训练记录的爬取；在线数据主要来源于用户在网站上操作（构建知识库、提交题解、回复问题等），需要通过埋点日志记录并通过消息队列传递给 Flink 进行消费。下面介绍数据爬取模块时的技术选型和选取原因。

4.2.1 编程语言 Golang

Golang（简称 Go）是由 Google 公司的 Robert Griesemer, Rob Pike 和 Ken Thompson 在 2009 年开源的一种编程语言，并于 2012 年 3 月发布了第一个稳定版本。

为什么本文选择 Go 作为系统开发的主要语言？

1. 性能优势：简单是 Go 语言的核心，这意味着它可以快速编译，快速运行并且快速上手开发。实测中 Go 拥有着近乎 C++ 的编译速度与 Python 的开发效率。

2. 并发：独特的并发设计是 Go 区别其他语言的最大特点。Go 的并发哲学源自 CSP 理论^[23]，选择 Channel 通信而不是加锁来控制共享资源，从设计上避免了数据竞争的问题。而并发的基本单位协程 Goroutine 相比线程来说，避免了操作系统内核对线程的调度，减少了大量 CPU 用来保存上下文和恢复现场的时间，对操作系统完全透明。这也是 Go 在编写 HTTP 服务器方面表现出色的主要原因之一。

3. 跨平台且易于维护：Go 编译生成的二进制文件可以在 Windows, Linux, macOS 或其他平台上直接运行，降低了部署的难度。

4.2.2 爬虫框架 Go-Colly

由于整个服务端由 Go 语言构建，便没有选择知名的 Python 爬虫框架 Scrapy，而是采用了 Go 语言的 Colly 爬虫框架。下表 2-1 展示了两者的差异：

表 4-1 爬虫框架 Colly 与 Scrapy 对比

对比内容	Colly	Scrapy
爬取速度	较快	较慢
API 设计	简洁	每个版本都有改变
并发支持	原生支持	需要配置
Robots 协议支持	有	有
分布式支持	支持	需要配置
解析库支持	Goquery	Pyquery / BeautifulSoup 等
定时任务	不支持	支持
扩展性	不支持	有许多第三方扩展

相较于 Scrapy 而言，作为一个新生爬虫框架，依靠着 Go 语言强大的并发性能和简洁的语言风格，在爬取速度和使用难易程度上都有一定优势。但是在可扩展性和爬取后数据的分发，都和生态完整的 Scrapy 框架有着一定的差距。

4.2.3 定时任务 Cron

由于 Go-Colly 框架不支持定时任务，结合笔者三台服务器被挖矿木马入侵时 crontab 文件被异常写入的经历，发现了 Go-Cron 库。其实现了类似于 Linux 系统 Crontab 服务的功能，利用这一功能完成了自动化定时爬取。

使用 Cron 不仅仅能够实现定时任务，还支持为定时任务自定义许多其他功能，让爬虫服务变成更加稳定以及便于维护。下表 2-2 列举了这些功能和实现意义：

表 4-2 Cron 自定义功能及其意义

功能	意义
WithLocation	指定时区
WithParser	使用自定义的解析器
WithLogger	自定义 log 来记录任务运行情况
ThreadSafe	显式的处理回调
WithChain	自定义 JobWrapper，可以用来捕获异常、恢复上次未执行完的 Job、跳过失败任务等

4.2.4 日志收集 Zap

Go 官方提供的日志库 Go Logger 虽然使用起来非常的简单，但是仅支持基本的日志事件，不支持输出 INFO、DEBUG 等多个日志等级。对于错误事件也仅仅是在抛出 panic 之前记录一条日志，缺少 ERROR 日志等级，无法在不退出程序的情况下记录错误。同时输出的日志不支持格式化，对接下来的数据处理和分析造成影响。所以选择了开源日志库 Zap 来对用户

产生的点击事件进行记录。Zap 针对语言底层，实现了一个无反射、零分配的 JSON 编码器，减少了在序列化数据时造成的时间开销。下表 2-3 展示了 Zap 与其他 Go 语言日志库的性能差距：

表 4-3 Zap 与其他日志库性能对比

Package	Time	Time % to zap	Objects Allocated
zap	862 ns/op	+0%	5 allocs/op
zerolog	4021 ns/op	+366%	76 allocs/op
go-kit	4542 ns/op	+427%	105 allocs/op
apex/log	26785 ns/op	+3007%	115 allocs/op
logrus	29501 ns/op	+3322%	125 allocs/op
log15	29906 ns/op	+3369%	122 allocs/op

从上表可以看出，Zap 极大的减少了每次操作所需要的资源分配，比另一个热门日志库 logrus 性能提升了 33 倍。在实用性方面，Zap 也弥补了上述所提到的官方库缺点，让我们可以随意的定制所需要的日志输出。

4.3 大数据特征处理模块

随之收集而来的数据越来越多，单个服务器可能无法保证数据的可靠存储。大数据技术是整个系统数据存储、清洗与训练中不可缺少的一环。本次设计采用的大数据技术相互之间的联系如下：

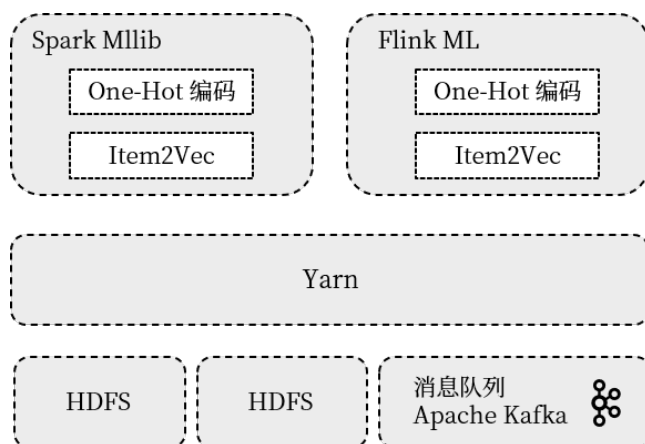


图 4-2 大数据技术关系图

如上图 2-1 所示：Hadoop HDFS 作为分布式文件存储，是所有落盘数据最终的去处。Yarn 作为资源调度和任务分配的中心对 Spark 的任务进行管理。而 Spark Mllib 分布式并行处理特征数据，并将生成的 Embedding 传给 TensorFlow 完成模型训练。

4.3.1 分布式文件系统 HDFS

由于系统中主要使用 HDFS 作为分布式文件存储，而 MapReduce 的任务被 Spark 取代，在此主要介绍以下 HDFS 实现高可用的原理。

HDFS 从设计之初就假设存储节点并不可靠，并设计了心跳检测、数据冗余备份等机制来帮助节点在发生故障时及时发现并恢复。**HDFS 的主要组件包括一个 namenode 和其管理的多个 datanode，以及 Secondarynamenode 检测节点。**

1. Namenode(NN): HDFS 的唯一主节点。主节点可以跟踪文件，管理文件系统，并在其中具有所有已存储数据的元数据。并直接与客户端联系。

2. Secondarynamenode(SN): 名称节点的检查节点。指示名称节点创建并向其发送镜像文件和操作日志，然后由 SN 创建压缩镜像。当主节点出现故障时，SN 会利用备份的镜像文件帮助其快速恢复。

3. Datanode (DN): 数据节点。每 3 秒向主节点报告一次，超过 2 分钟将被视为掉线，这时由其他节点上存储的备份数据会重新构建一个 DN。

下图 2-2 展示了 HDFS 的组织架构：

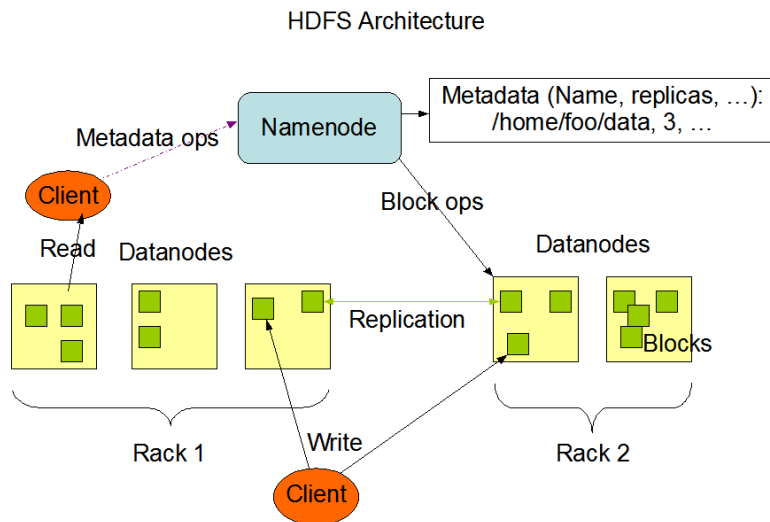


图 4-3 HDFS 架构图

4.3.2 分布式计算平台 Spark

Spark 生态系统如下图 2-3 所示：

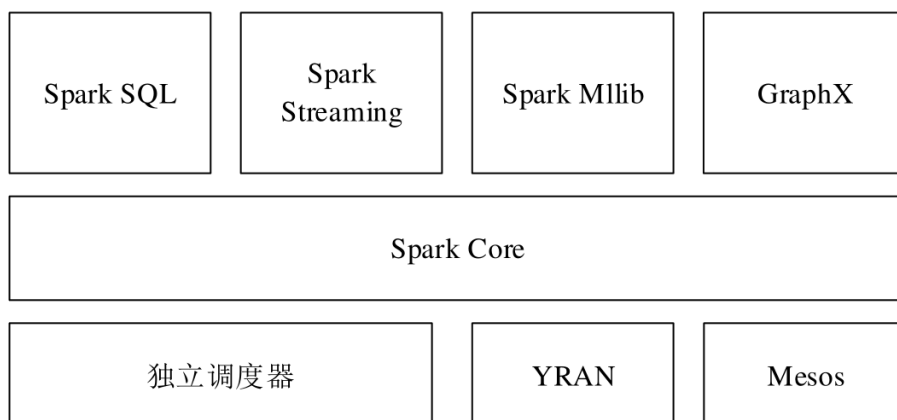


图 4-4 Spark 生态系统

Spark 是一个面向内存的计算框架，这使得它的计算效率远远超过 MR 任务。而 SparkSQL 更是在此基础上抽象出 DataFrame 数据结构，并为其做了大量的性能优化，使得 Python/Java 等语言有了等同于原生 scala 编程的执行效率。

下表 2-4 列出了 Spark 各个模块的功能：

表 4-4 Spark 各模块功能表

模块名	功能
Spark Core	Spark Core 定义操作 RDD 的相关 API，实现内存管理、错误恢复、任务调度、与存储系统交互等基础功能。
Spark SQL	Spark SQL 用于操作各种结构化数据，通过 Spark SQL，可以使用 SQL 来查询或修改数据。
Spark Streaming	Spark Streaming 作为流式计算组件，定义用于操作实时数据流的相关 API。
Spark MLlib	Spark MLlib 定义一些常见的机器学习相关算法和功能，如分类、聚类、回归分析、协同过滤等，同时支持数据导入、模型评估等额外功能。
集群管理器	集群管理器 负责整个集群的资源管理以及节点之间的通信，从而灵活地实现集群规模的伸缩扩展。Spark 支持在 Hadoop YARN、Apache Mesos 等多种集群管理器上部署和运行。

4.4 推荐模块

4.4.1 模型训练 TensorFlow

4.4.2 模型线上服务 TensorFlow Serving

4.5 系统业务模块

4.5.1 前端框架 Vue

4.5.2 后端框架 Gin

4.6 本章小结

本章主要建立了……
，给出了……，对……进行了推导……

第5章 程序设计竞赛组队推荐系统实现

5.1 环境部署

5.1.1 Hadoop 集群部署

5.1.2 Spark 集群部署

5.1.3 Flink 集群部署

5.2 算法实现与结果分析

5.3 系统实现

5.3.1 数据收集模块实现

5.4 本章小结

结论

[单击此处输入结论]

致谢

[单击此处输入致谢]

参考文献

- [1] 中国互联网络信息中心(CNNIC). 第 47 次中国互联网络发展现状统计报告[R]. 2021 年 2 月 3 日.
- [2] 周晓琳,解静,刘勇,尤枫,吴佳伟.面向程序设计竞赛的人才培养模式实践与探索[J].大学教育,2020(07):144-146.
- [3] GOLDBERG D, NICHOLS D, OKI B M. Using collaborative filtering to weave an information tapestry[J]. Communications of the ACM, 1992, 35(12): 61–70. DOI:10.1145/138859.138867.
- [4] GREG L, BRENT S, JEREMY Y. Industry report: Amazon.com recommendations: Item-to-item collaborative filtering[J]. IEEE Distributed Systems, 2003: 76–80.
- [5] Gai K, Zhu X, Li H, et al. Learning piece-wise linear models from large scale data for ad click prediction[J]. arXiv preprint arXiv:1704.05194, 2017.
- [6] Rendle S. Factorization machines[C]//2010 IEEE International Conference on Data Mining. IEEE, 2010: 995-1000.
- [7] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Advances in neural information processing systems, 2012, 25: 1097-1105.
- [8] Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[J]. arXiv preprint arXiv:1310.4546, 2013.
- [9] Barkan O, Koenigstein N. Item2vec: neural item embedding for collaborative filtering[C]//2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP). IEEE, 2016: 1-6.
- [10] Grover A, Leskovec J. node2vec: Scalable feature learning for networks[C]//Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. 2016: 855-864.
- [11] Wang J, Huang P, Zhao H, et al. Billion-scale commodity embedding for e-commerce recommendation in alibaba[C]//Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018: 839-848.
- [12] 张廉月. 基于 Flink 的电影推荐系统的研究与实现[D]. 电子科技大学, 2020.
- [13] 景坤. 网约车竞赛平台的设计与实现[D]. 北京交通大学, 2019.
- [14] 胡开冉. 基于 Hadoop/Spark 的反向推荐算法研究[D]. 成都理工大学, 2018.
- [15] 李斌. 面向学科竞赛的组队平台研究[D]. 华中师范大学, 2020.
- [16] Toledo R Y, Mota Y C. An e-learning collaborative filtering approach to

- suggest problems to solve in programming online judges[J]. International Journal of Distance Education Technologies (IJDET), 2014, 12(2): 51-65.
- [17] Yera Toledo R, Caballero Mota Y, Martínez L. A recommender system for programming online judges using fuzzy information modeling[C]//Informatics. Multidisciplinary Digital Publishing Institute, 2018, 5(2): 17.
- [18] Caro-Martinez M, Jimenez-Diaz G. Similar users or similar items? Comparing similarity-based approaches for recommender systems in online judges[C]//International Conference on Case-Based Reasoning. Springer, Cham, 2017: 92-107.
- [19] FANTOZZI P, LAURA L. Recommending Tasks in Online Judges using Autoencoder Neural Networks[J]. 2020.
- [20] 朱国进, 凌晓晨. 基于关联规则挖掘的 OJ 推荐方法[J]. 收藏, 2018, 2.
- [21] 孙权, 贺细平. 协同过滤算法在 ACM 在线评测推荐系统中的应用研究[J]. 电脑与信息技术, 2015, 6.
- [22] 肖春芸, 贺樛, 窦亮, 等. 基于 Online Judge 进行程序设计类课程实践的混合推荐[J]. 现代计算机, 2019.
- [23] Hoare C A R. Communicating sequential processes[J]. Communications of the ACM, 1978, 21(8): 666-677.

附录 A

附录 B

附录 C