**哈 尔 滨 理 工 大 学**

**毕 业 设 计**

**题 目：程序设计竞赛组队推荐系统研究与设计**

**院、 系： 计算机科学与技术学院 计算机系**

**姓 名： 鲁明翰**

**指导教师： 高明**

**系 主 任： 孙冬璞**

**2021 年 6 月 11 日**

**哈尔滨理工大学毕业设计（论文）评语**

|  |
| --- |
| 学生姓名： 鲁明翰 学号：1704010913 |
| 学 院：计算机科学与技术学院 专业：计算机科学与技术专业 |
| 任务起止时间： 2021年 3 月 1 日 至 2021 年 6 月 11 日 |
| 毕业设计（论文）题目：程序设计竞赛组队推荐系统研究与设计 |
| 指导教师对毕业设计（论文）的评语：  指导教师签名： 指导教师职称： |
| 评阅教师对毕业设计（论文）的评语：  评阅教师签名： 评阅教师职称： |
| 答辩委员会对毕业设计（论文）的评语：  答辩委员会评定，该生毕业设计（论文）成绩为：  答辩委员会主席签名： 职称：  年 月 日 |

教务处制表

**哈尔滨理工大学毕业设计（论文）任务书**

|  |  |
| --- | --- |
| 学生姓名：鲁明翰 学号：1704010913 | |
| 学 院：计算机科学与技术 专业：计算机科学与技术 | |
| 任务起止时间： 2021年 3 月 1 日至 2021 年 6 月 11日 | |
| 毕业设计（论文）题目：程序设计竞赛组队推荐系统研究与设计 | |
| 毕业设计工作内容：  研究和实现一个高可用的在线推荐系统。从经典的机器学习算法到目前火爆的深度学习模型，寻找一种适合程序设计竞赛组队的推荐模型，并对其流程进行修改，使用推荐系统来扩充整个队伍知识的广度。  进度安排：  2021.3-2021.4：完成推荐算法理论的学习；  2021.4-2021.4：学习Spark、Docker、TensorFlow等平台的使用；  2021.4-2021.5：学习使用Go语言完成Web Server的搭建并完成前端页面；  2021.5-2021.6：整理资料完成论文。 | |
| 资料：  1. 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.  2. 景坤. 网约车竞赛平台的设计与实现[D]. 北京交通大学, 2019.  3. 李斌. 面向学科竞赛的组队平台研究[D]. 华中师范大学, 2020.  4. 王喆. 深度学习推荐系统[M].北京：电子工业出版社，2020.3. | |
| 指导教师意见：  签名：  年 月 日 | 系主任意见：  签名：  年 月 日 |

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Faiss实现ANN近邻搜索并提供gRPC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](#_Toc74123207)

[1.1 研究背景及意义 1](#_Toc74123208)

[1.2 国内外研究与发展现状 2](#_Toc74123209)

[1.2.1 推荐系统研究与发展现状 2](#_Toc74123210)

[1.2.2 推荐算法在算法竞赛领域的研究现状 4](#_Toc74123211)

[1.3 论文的主要工作 5](#_Toc74123212)

[1.4 论文的组织结构 5](#_Toc74123213)

[第2章 推荐算法的研究与改进 6](#_Toc74123214)

[2.1 传统推荐算法理论研究 6](#_Toc74123215)

[2.1.1 基于内容的推荐算法 6](#_Toc74123216)

[2.1.2 常用向量相似度计算方法 7](#_Toc74123217)

[2.1.3 基于协同过滤的推荐算法 7](#_Toc74123218)

[2.1.4 协同过滤的优化—矩阵分解 9](#_Toc74123219)

[2.2 Embedding技术对推荐算法的改进 11](#_Toc74123220)

[2.2.1 Embedding向量的含义与运用方法 11](#_Toc74123221)

[2.2.2 Embedding向量的生成方法 11](#_Toc74123222)

[2.2.3 Embedding向量的快速检索 14](#_Toc74123223)

[2.3 深度学习在推荐系统中的应用 15](#_Toc74123224)

[2.4 本章小结 16](#_Toc74123225)

[第3章 程序设计竞赛组队推荐系统分析 17](#_Toc74123226)

[3.1 系统需求分析 17](#_Toc74123227)

[3.1.1 非功能性需求 18](#_Toc74123228)

[3.2 开发环境分析 19](#_Toc74123229)

[3.2.1 硬件环境 19](#_Toc74123230)

[3.2.2 软件环境 19](#_Toc74123231)

[3.3 本章小结 19](#_Toc74123232)

[第4章 程序设计竞赛组队推荐系统设计 20](#_Toc74123233)

[4.1 系统概要设计 20](#_Toc74123234)

[4.1.1 功能模块划分 22](#_Toc74123235)

[4.1.2 接口数据定义 22](#_Toc74123236)

[4.1.3 技术路线选择 24](#_Toc74123237)

[4.2 系统详细设计 26](#_Toc74123238)

[4.2.1 数据收集模块详细设计 26](#_Toc74123239)

[4.2.2 大数据特征处理模块详细设计 29](#_Toc74123240)

[4.2.3 推荐模块详细设计 33](#_Toc74123241)

[4.2.4 系统业务模块详细设计 37](#_Toc74123242)

[4.3 本章小结 40](#_Toc74123243)

[第5章 程序设计竞赛组队推荐系统实现 41](#_Toc74123244)

[5.1 环境部署 41](#_Toc74123245)

[5.1.1 Hadoop集群部署 41](#_Toc74123246)

[5.1.2 Spark集群部署 42](#_Toc74123247)

[5.2 系统实现 42](#_Toc74123248)

[5.2.1 数据收集模块实现 42](#_Toc74123249)

[5.2.2 特征处理模块实现 44](#_Toc74123250)

[5.2.3 推荐模块实现 46](#_Toc74123251)

[5.2.4 系统业务模块实现 47](#_Toc74123252)

[5.3 本章小结 49](#_Toc74123253)

[结论 50](#_Toc74123254)

[致谢 51](#_Toc74123255)

[参考文献 52](#_Toc74123256)

[附录A 54](#_Toc74123257)

[附录B 59](#_Toc74123258)

[附录C 63](#_Toc74123259)

# 绪论

## 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Hadoop可以使用多台普通的机器组成一个集群，它的三个核心组件HDFS、Yarn、MapReduce构成了大数据处理的三驾马车。其中HDFS提供了分布式存储，MapReduce在HDFS的基础上实现并行计算，Yarn负责管理和调度整个集群的资源。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，让每一届学生们的知识汇聚于此，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。

## 国内外研究与发展现状

### 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，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研究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n2 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，往往冷门物品得不到很好的推荐。

2006年，矩阵分解算法在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）[4]，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。传统推荐模型三大模型的演化关系如图1-1所示：

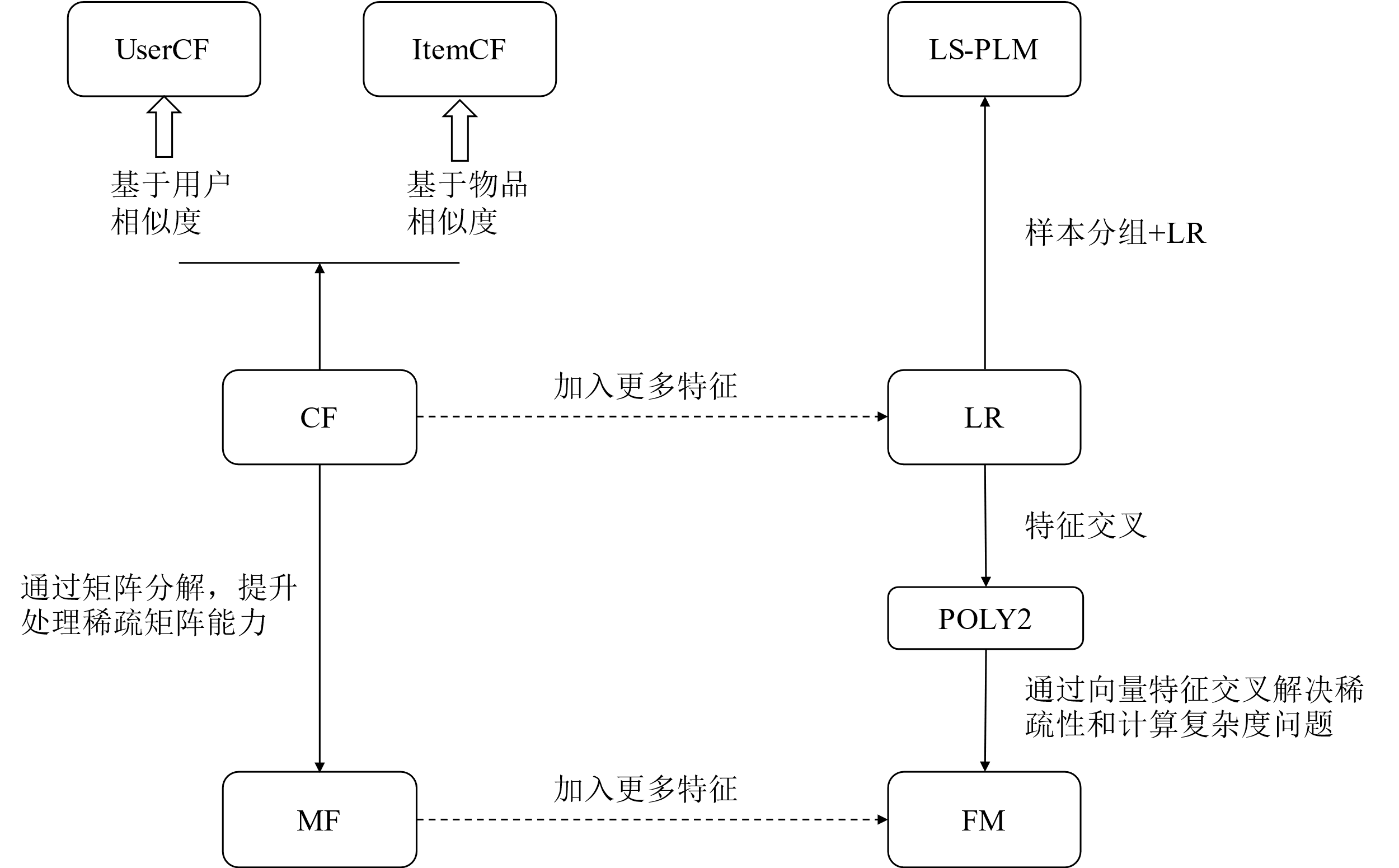


图 1‑1 传统推荐模型演化关系图

随着Alex Krizhevsky在2012年提出了AlexNet，开启了深度学习的时代，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[5]。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[6]。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[7]，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[8]。

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[9]。景坤使用了推荐系统完成了网约车组队的设计与实现[10]，胡开冉提出了基于FunkSVD 的反向推荐可以大大提高物品的覆盖率[11]，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[12]。

### 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[13]。Year等人在2018年提出基于模糊逻辑的方法[14]。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[15]。2020年fantozzi等人使用了欧洲学生在意大利信息学奥利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[16]。

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[17]，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[18]，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[19]。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。

## 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。

## 论文的组织结构

本文一共分为五章，下面对每章的内容进行简要描述：

第一章为绪论，对整个论文研究的背景和意义进行了介绍，接着分析了推荐系统国内外的研究现状，梳理了其发展的路线。最后对系统所完成的主要功能进行了说明。

第二章为推荐算法的研究和改进。首先介绍了几种传统推荐算法的原理以及它们的不足，然后应用Embedding技术对其不足进行了改进。

第三章为系统分析。首先对系统进行需求分析，对最终实现的功能范围做出了约束。然后分析了系统实现所需要的软硬件环境，为接下来的设计与实现打下了坚实的基础。

第四章为系统设计。首先对系统进行概要设计，进行了模块间的功能划分、接口设计以及技术路线选择，然后对各个模块的详细设计进行了介绍。

第五章为系统实现。主要展示了系统各个模块所实现的功能。并对推荐模型的效果做出了评估。

# 推荐算法的研究与改进

## 传统推荐算法理论研究

### 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：

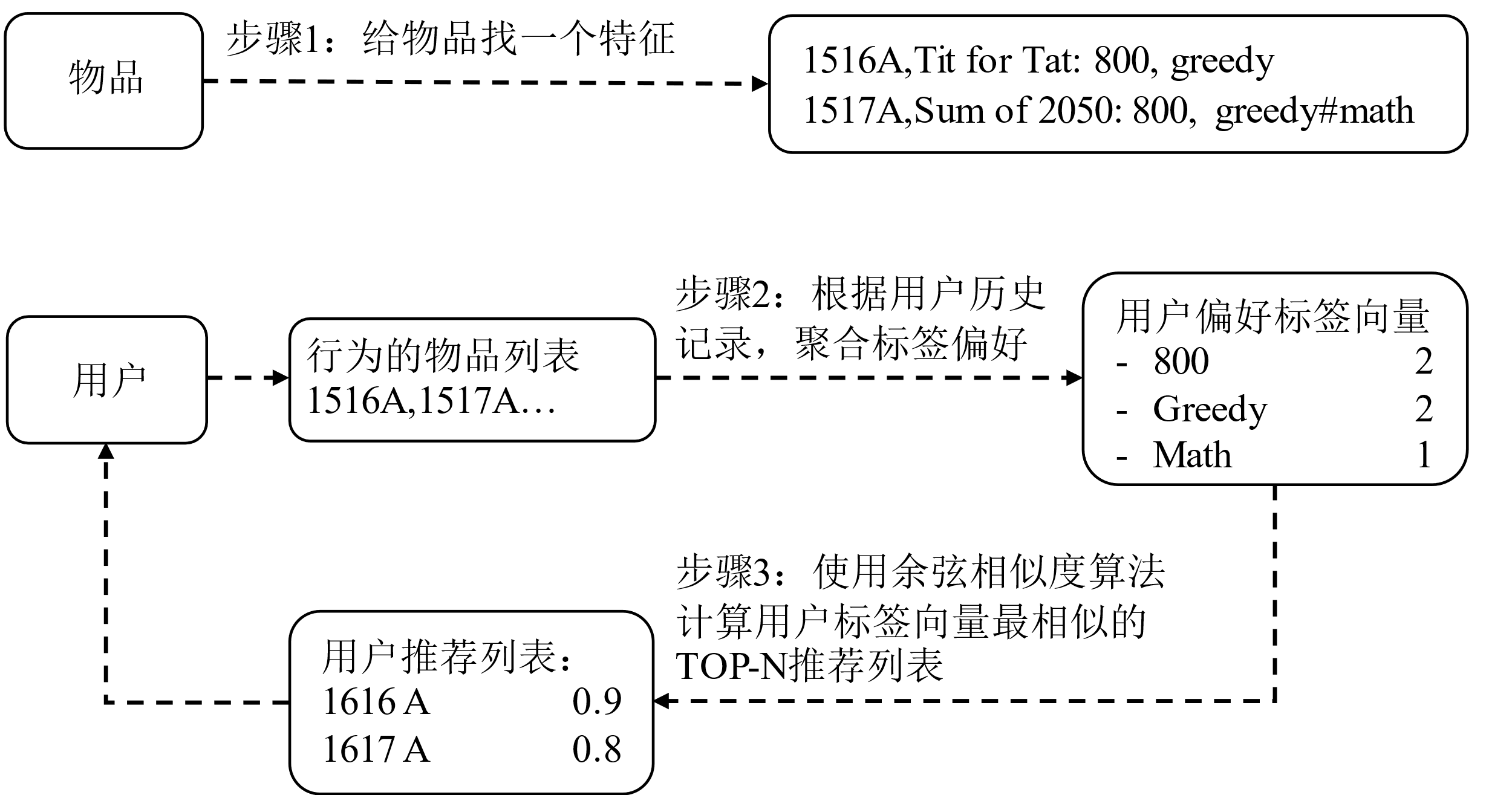


图 2‑1 基于内容推荐算法流程

**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2-1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 |

### 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。

**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。

**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。

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。

### 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。

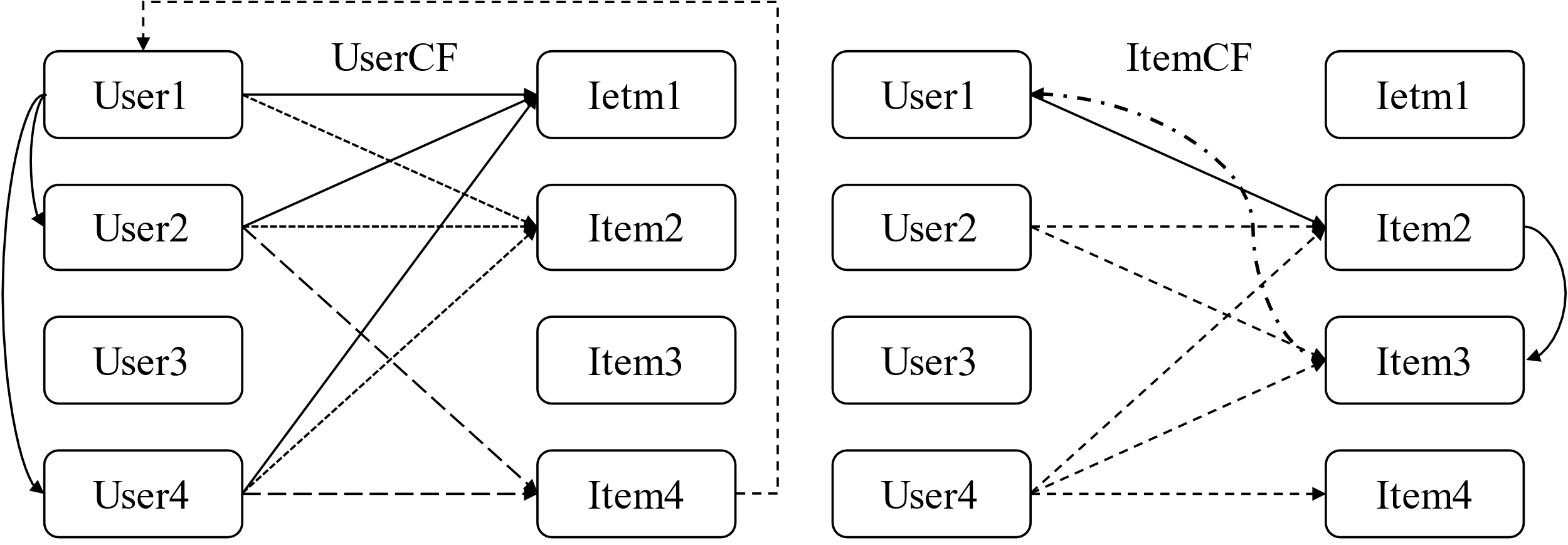


图 2‑2 UserCF与ItemCF原理

在获得了Top-N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。整个过程如（式2-5）所示，

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User3，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。

下面对算法流程进行描述：

**1.** 第一步 获得用户的历史行为数据，组成维的用户-物品矩阵

**2.** 第二步 以列为向量，通过皮尔逊相关系数计算物品之间的相似度，最终构建出维的物品相似度矩阵。

**3.** 第三步 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.** 第四步 针对目标用户的喜爱列表，分别每个对已评价物品生成个相似物品，组成相似物品候选集。

**5.** 第五步 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：

其中是目标用户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，权重是未评分物品p与已评分物品h的相似度，是用户u对h的评分。

下表2-2展示了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 | 基于用户相似度进行推荐，有着更强的社交属性。 | 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。 |
| 适用于发现热点，以及跟踪热点的发展趋势。 | 用户相似度矩阵的维护非常消耗资源，在线存储系统难易承受。 |
| ItemCF | 对冷启动优化更好，更适用于兴趣变化较稳定的应用 | 泛化能力较差，冷门的物品会很少被推荐到 |

### 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：

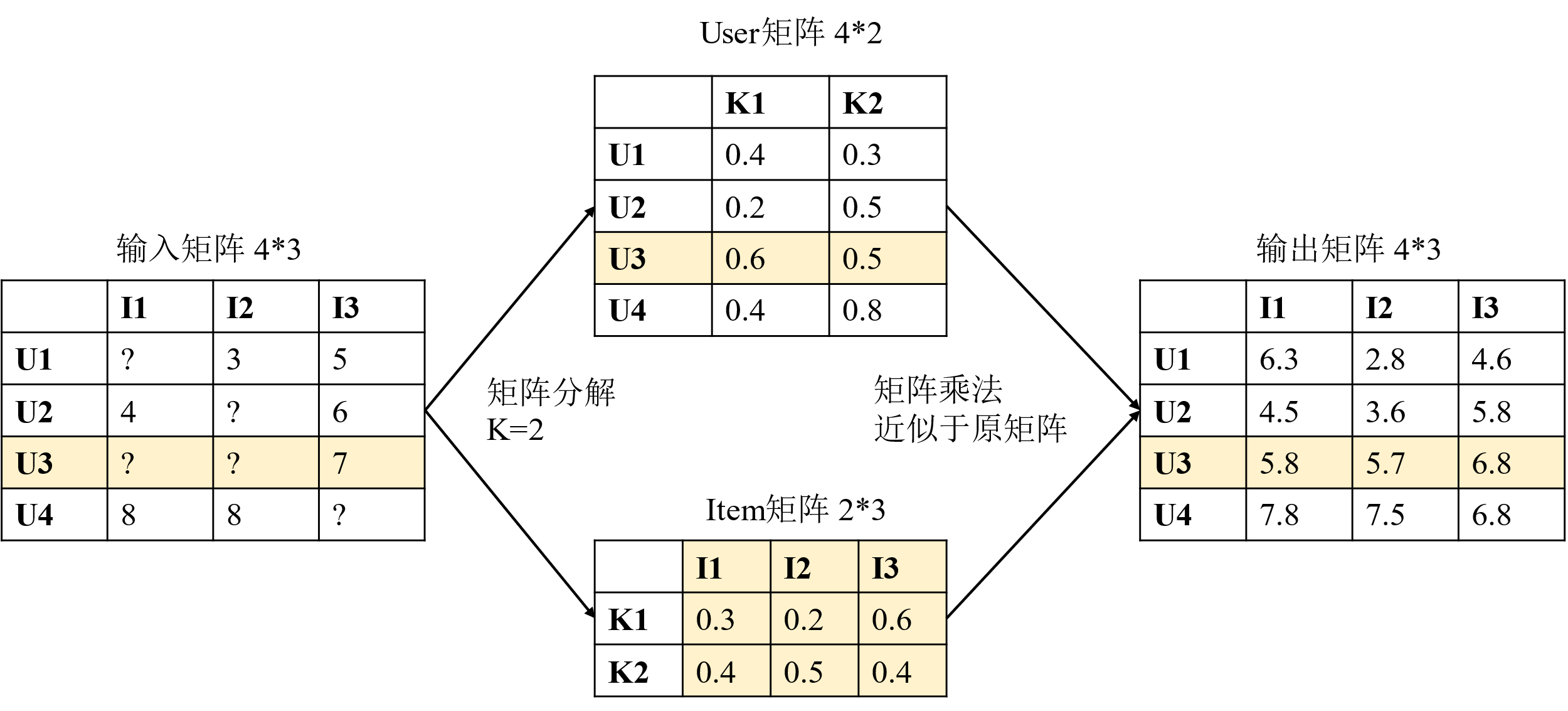


图 2‑3 矩阵分解原理

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矩阵为维，Item矩阵为维，K表示特征的数量。分解所得到的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，而我们可以通过多轮训练，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。

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：其中表示物品i在Item矩阵中所对应的列向量，表示用户u在User矩阵中对应的行向量。

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。

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： 为正则化系数，代表约束模型稳定性的能力。

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 求偏导的结果如（式2-10）所示：

对求偏导的结果如（式2-11）所示：

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：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：

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 |

## Embedding技术对推荐算法的改进

### Embedding向量的含义与运用方法

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。

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 |

### Embedding向量的生成方法

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Word2Vec出现之前，人们一直使用One-Hot编码进行向量化操作。

One-Hot编码将词典集合中的词映射为一个维向量，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 0 |
| Love | 0 1 0 0 0 |
| ICPC | 0 0 1 0 0 |
| Like | 0 0 0 1 0 |
| Code | 0 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维稀疏One-Hot向量映射到高维度空间中长度为维的一个点。这里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：

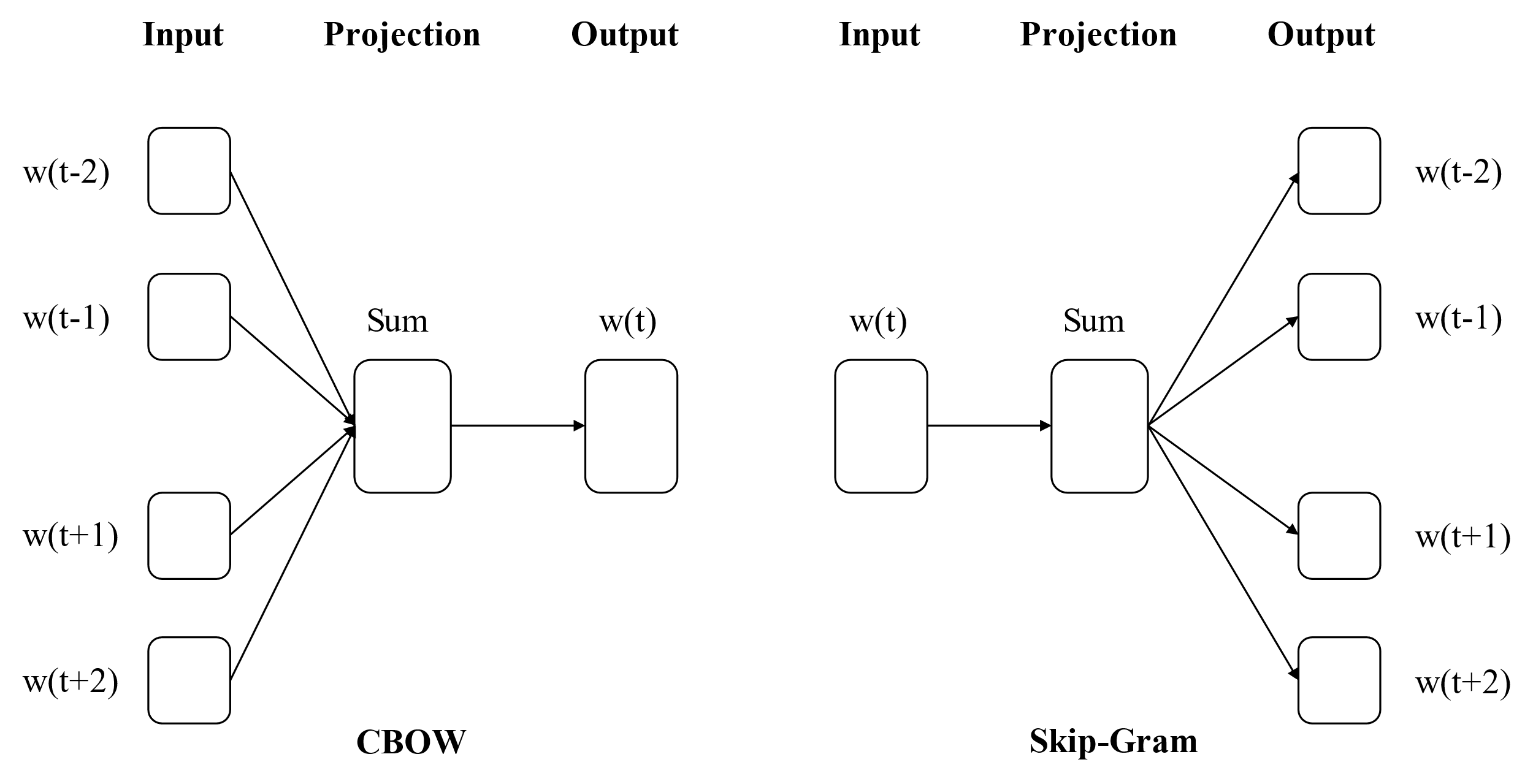


图 2‑4 CBOW模型与Skip-Gram模型

这两种模型的区别就在于，CBOW模型是通过中心词周围的词来预测中心词；Skip-gram模型是通过中心词来预测周围词。

CBOW简化版的模型（仅输入一个词）如下图2-5所示：

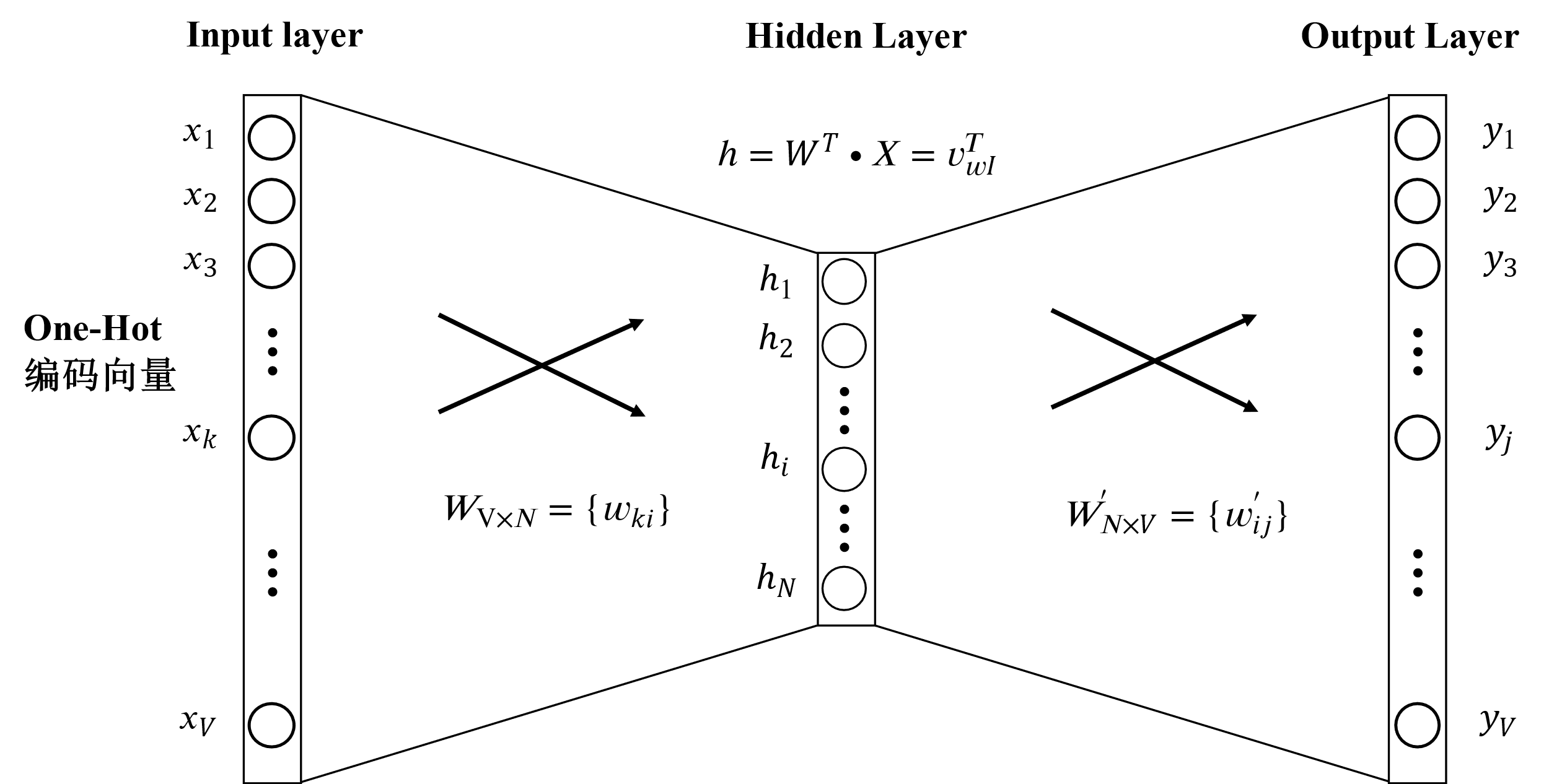


图 2‑5 CBOW简化版模型

**1.** 输入层约定输入的词库大小为，中间的隐层大小为。若将输入单词标记为，则输入层是其经过One-hot编码后的向量表示，一共有个神经元，除了第个神经元为1，其余各项均为0，例如。

**2.** 权重矩阵输入层向量到隐层之间的权值可以用一个的矩阵表示，其每一行的值对应着一个我们最终需要的Embedding向量。将One-Hot编码过后的向量映射为一个N维的短向量，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：

**4.** 权重矩阵 在隐层到输出层之间有另一个权重矩阵。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。其公式如（式2-15）所示：

其中表示权重矩阵的第j列。

**5.** 输出层在输出时将每一个词的得分通过softmax函数归一化得到一个在之间得概率，其表示为在输入词为时输出词是的概率。可以用（式2-16）对其进行表示：

将（式2-14、2-15）带入（式2-16）整理可得（式2-17）：其中和分别是输入词的两种不同的N维向量表示形式。

其优化的目标函数L如（式2-18）所示：

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：

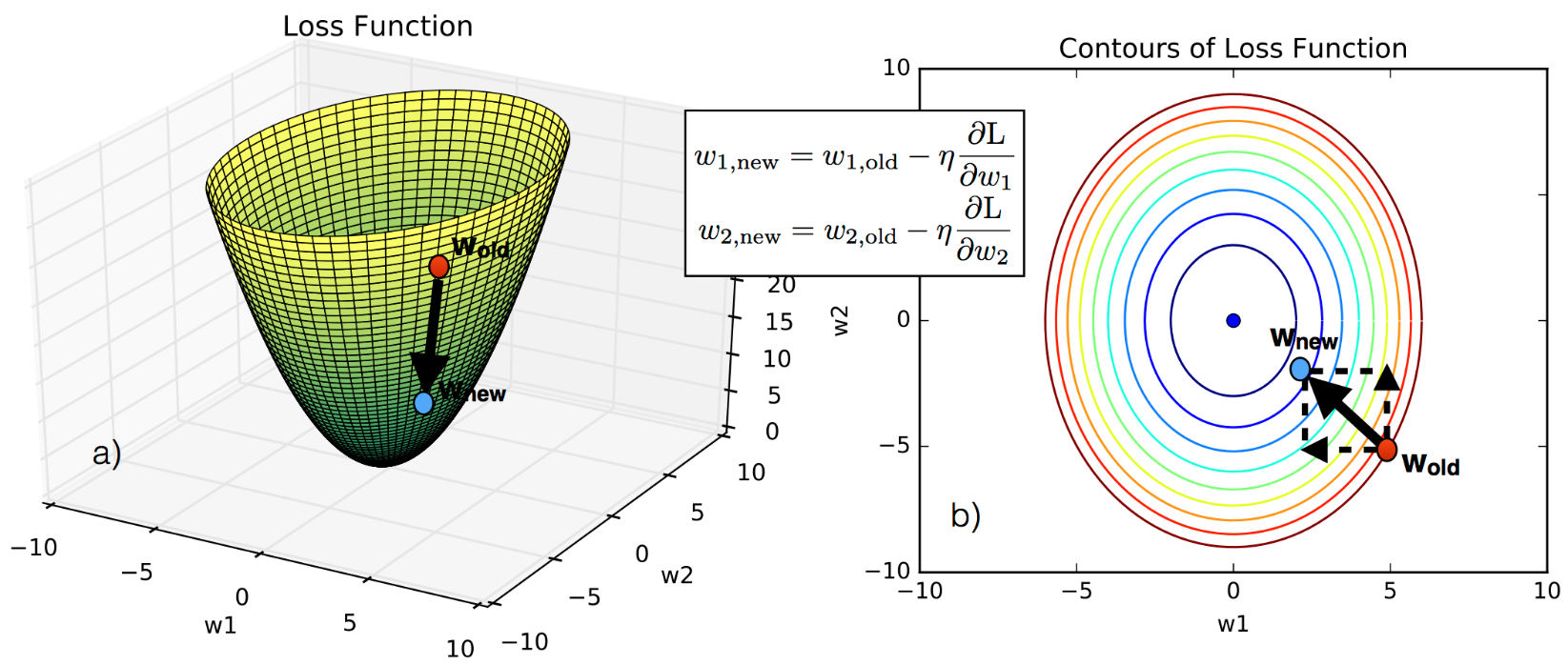


图 2‑6 Word2Vec梯度下降求最优权值

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序列上的Embedding向量，这就是Item2Vec算法。

### Embedding向量的快速检索

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：

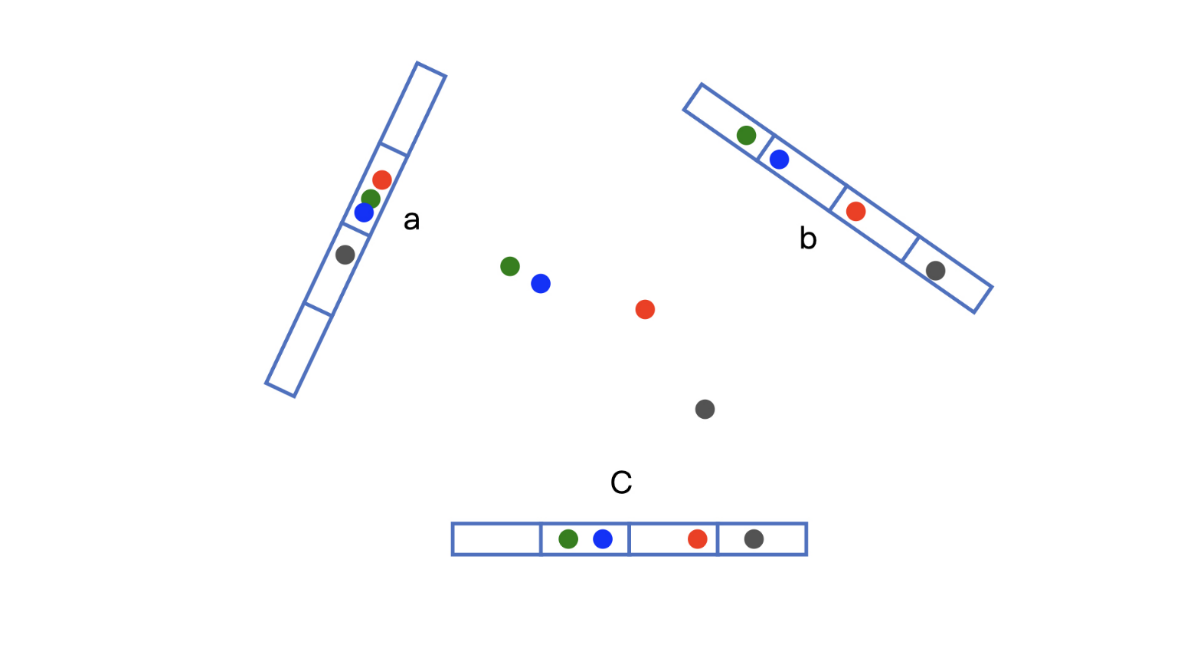


图 2‑7 高维空间点向低维空间映射

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。解决这个问题的方法是通过多个哈希函数将其进行分桶，如果两个点同时掉入多个哈希函数生成的桶中时，那么它们相似的概率就会大大提高，减少误判的情况。

## 深度学习在推荐系统中的应用

如果将矩阵分解看成一个简单的神经网络，那么其结构如图2-8所示：

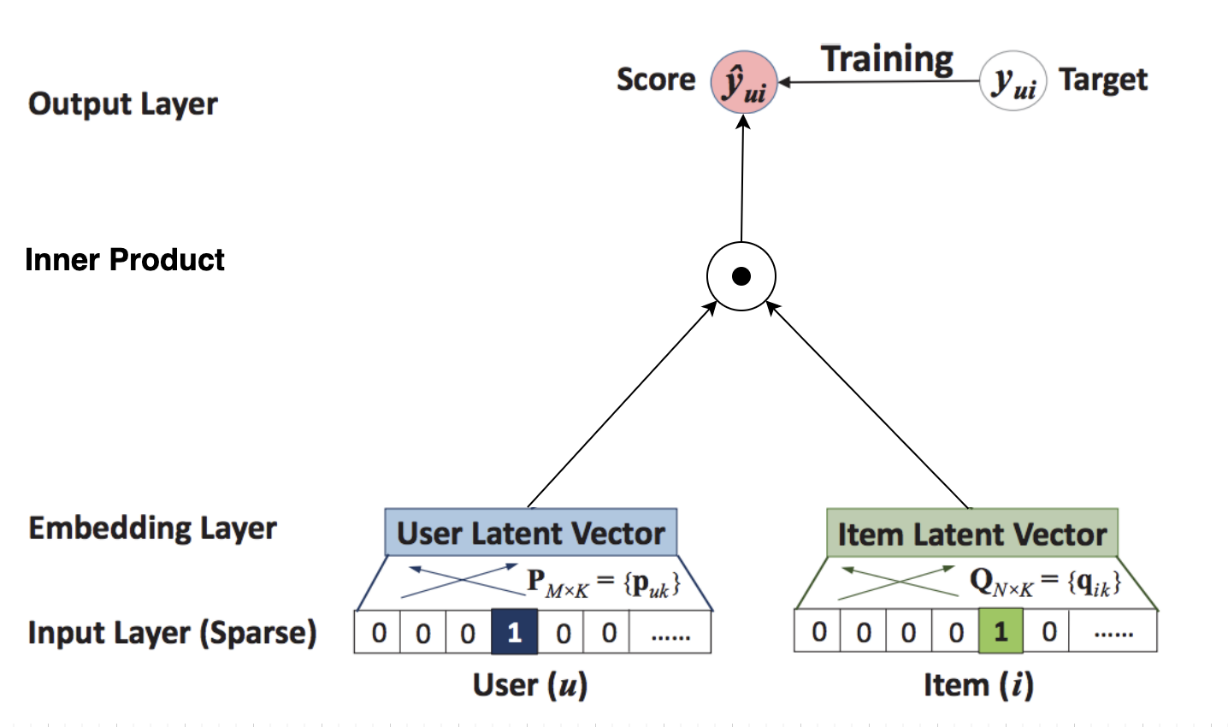


图 2‑8 矩阵分解的神经网络结构

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：

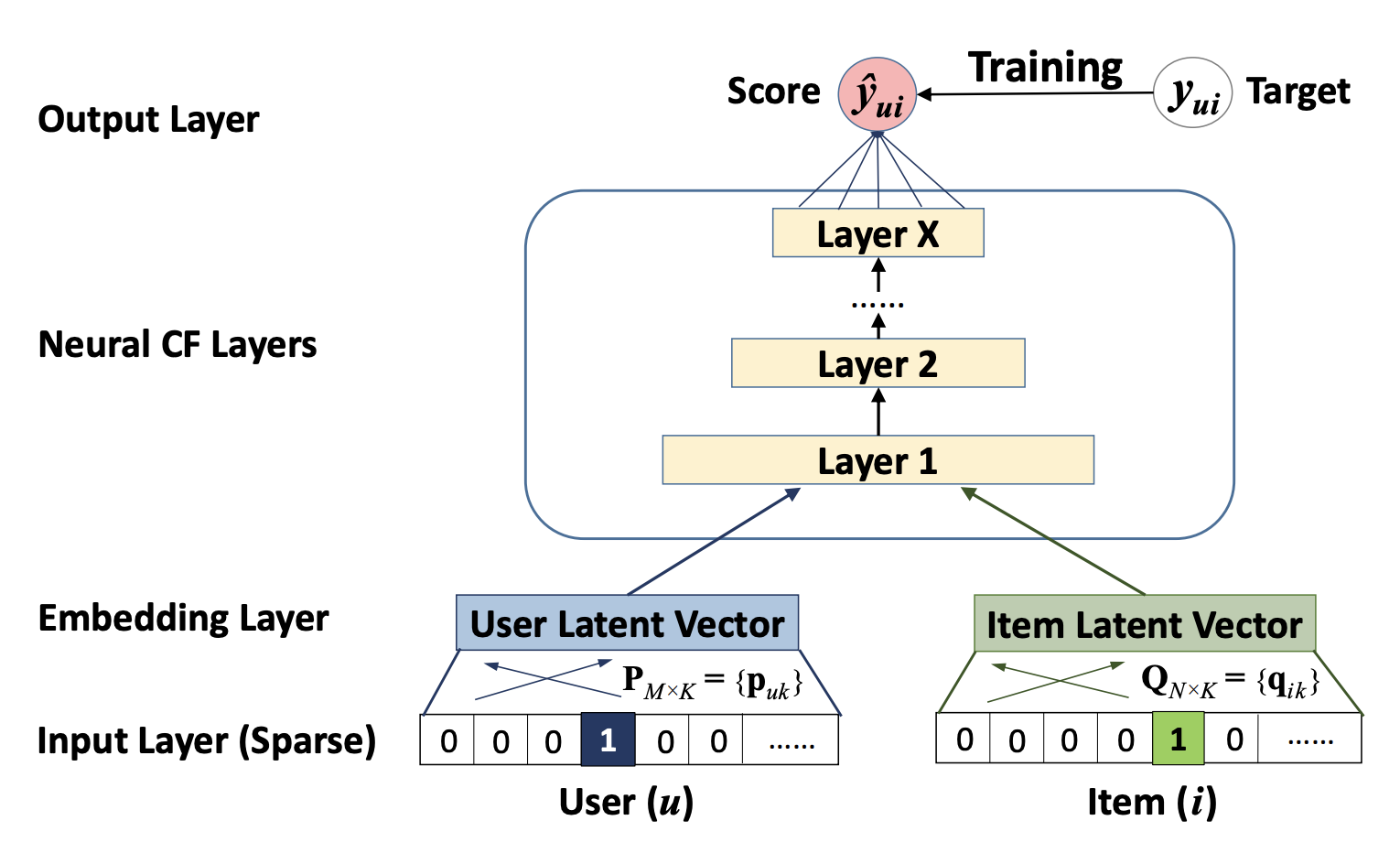


图 2‑9 NeaulCF神经网络结构

NeuralCF实际上在处理embedding特征时，采用了两种方法。一种是GMF（广义矩阵分解），对输入的向量进行简单的内积运算，另外一种是MLP（多层感知机）。利用多层神经网络对user和item的特征进行了充分交叉。

NeuralCF带来了一种可以将User和Item两部分特征分别处理，然后进行不同方式交叉的处理思路。双塔模型便是根据这种思想，进行了进一步的改进。本文最终使用双塔模型对Embedding进行生成，其改进思想和模型结构将在推荐系统详细设计阶段进行阐述。

## 本章小结

本章为推荐算法的研究和改进。首先介绍了基于内容、协同过滤、矩阵分解等算法的原理以及它们的不足，然后应用Embedding技术对其不足进行了改进。最后介绍了如何运用深度学习模型NeaulCF在特征交叉方面进行改进。

# 程序设计竞赛组队推荐系统分析

## 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。需求分析主要包括系统的功能分析和非功能方面的需求。进行详细而完善的需求分析是系统后续能够顺利设计的保障。

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最多的10道题目生成推荐列表 |
| 历史热门题目 | 将codefoces历史上通过次数最多的10道题目生成推荐列表 |

续表3-1

|  |  |  |
| --- | --- | --- |
| 系统模块 | 功能性需求 | 具体描述 |
| 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 |

### 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。

## 开发环境分析

### 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。

### 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 |

## 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。

# 程序设计竞赛组队推荐系统设计

## 系统概要设计

在程序设计竞赛组队推荐系统需求分析中，将本系统划分为4个模块。它们分别为数据收集模块、特征处理模块、推荐模块以及系统业务模块。系统的总体功能架构如下图4-1所示：

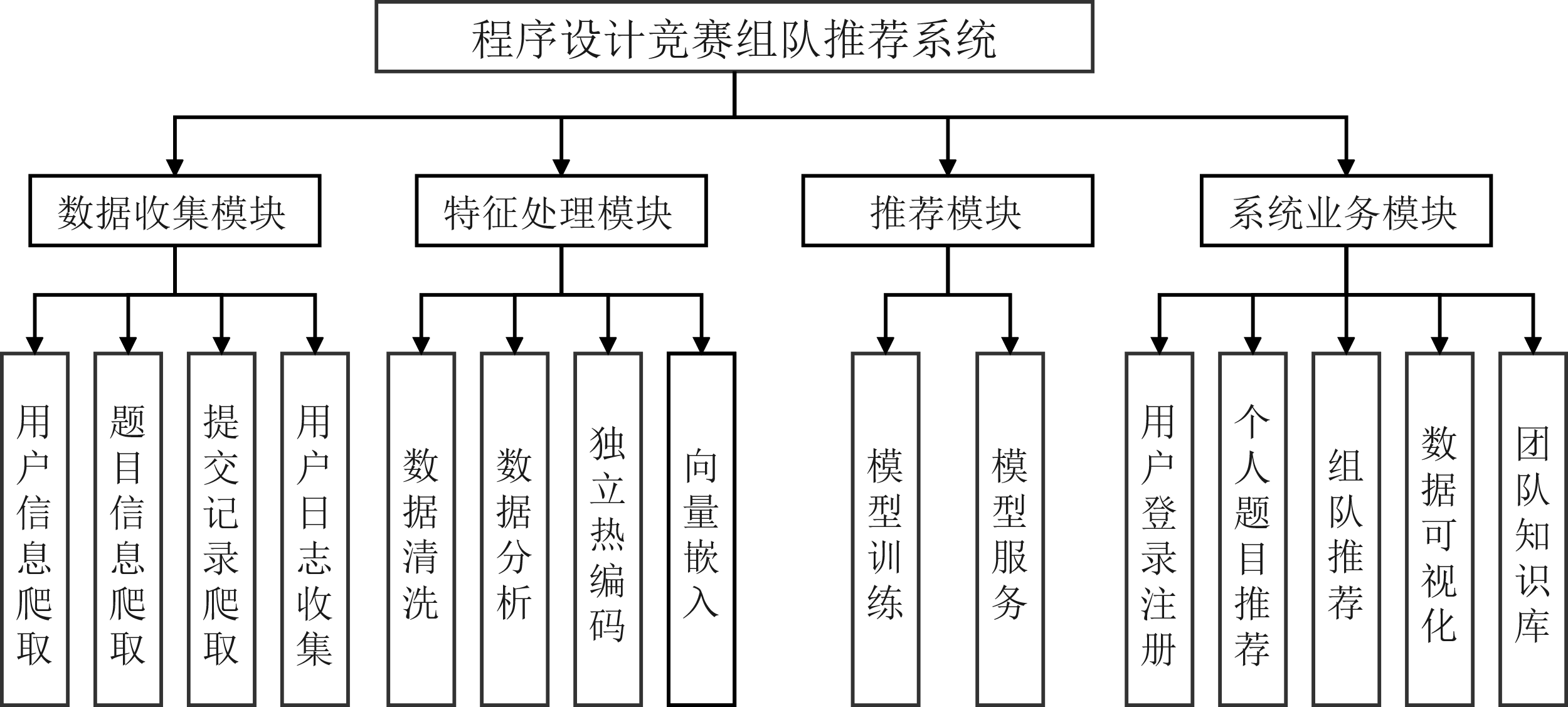


图 4‑1 系统的总体功能架构

其中，数据收集模块又包括用户信息、题目信息、用户提交记录等多种信息的爬取，以及用户在使用系统时所进行的重要操作收集（如对Wiki进行提交或修改）。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”。数据收集决定了接下来推荐模型训练效果的上限。

特征处理模块主要分为数据清洗、数据分析、One-hot编码和Embedding向量生成部分。将数据收集模块收集来的数据进行加工处理，来使接下来的模型训练更加的方便。该模块同时也对一些简单的用户属性进行了计算，如获得用户在每个题目类别下的平均分数。

推荐模块的功能比较单一，主要任务是构建NeaulCF推荐模型，将特征处理所得到的Embedding向量作为输入，利用多层神经网络将用户的特征充分交叉融合，最终得到每个用户的相似度TonN列表。并对外提供API接口调用，方便线上业务调取使用。

系统业务模块负责处理用户交互的信息。将得到的推荐列表和各种可视化图标进行展示，并维护一个Wiki库来进行知识的共享。

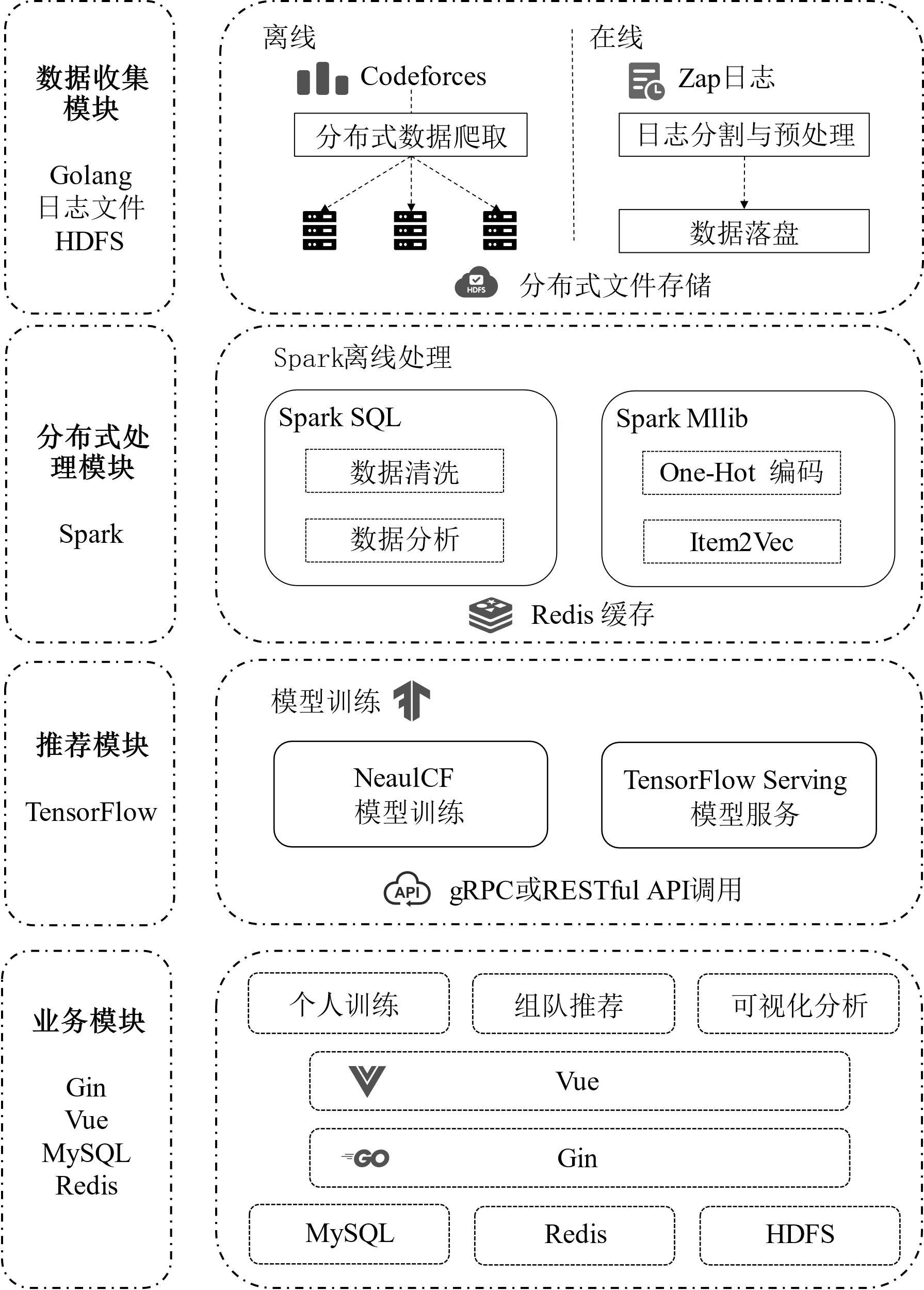


图 4‑2 总体技术架构

本系统的总体技术架构如上图4-2所示。下面会对系统的功能模块划分、技术路线选择以及数据库设计做详细说明。

### 功能模块划分

在经过需求分析之后，将整个系统划分为四个模块：数据收集模块、大数据特征处理模块、推荐模块以及系统业务模块。下表4-1对各个模块的功能进行了定义：

表 4‑1 各模块功能定义

|  |  |
| --- | --- |
| 模块 | 功能 |
| 数据收集模块 | 从Codeforces上爬取用户列表、题目列表、用户提交列表；使用日志记录用户问答情况和Wiki的提交情况。最终将全部数据保存到分布式存储HDFS中。 |
| 大数据特征处理模块 | 从HDFS中读取数据到Spark DataFrame中，进行数据预处理：删除缺失值、计算用户各个类别得分均值、对ID和类别信息进行One-hot处理，对历史行为使用Item2Vec生成Embedding向量。 |
| 推荐模块 | 将Spark训练得到的Embedding向量由TensorFlow训练双塔推荐模型，并通过TensorFlow Serving对在线服务器提供API调用，得到每个用户的推荐列表。 |
| 系统业务模块 | 前端页面主要实现用户的交互，将用户训练数据可视化展示。后端处理具体的业务流程，如用户的登录与注册、推荐列表生成、可视化图形生成、问答数据存储。 |

### 接口数据定义

划分模块之后需要对各个模块之间的数据传递接口做出约定，下面简要的对各个接口的数据形式和数据类型做出说明，更详细的物理结构设计将在各个模块的详细设计里面介绍。

数据收集模块的接口数据定义如下表4-2所示：

表 4‑2 数据收集模块接口定义

|  |  |  |
| --- | --- | --- |
| 接口描述 | 数据形式 | 数据类型 |
| 用户名输入接口 | [username1, username2,…] | 字符串 |
| 用户信息获取接口 | User[uid:1 ,score:2100 ,acNum:100] | 结构体数组 |
| 题目信息获取接口 | Problem[pid:1 ,score:2000 ,tag:graph…] | 结构体数组 |
| 提交列表获取接口 | Submission[uid:1, pid:1, actime: time] | 结构体数组 |
| 用户日志获取接口 | Log[uid: 1, optag: pr, time: time] | 结构体数组 |
| 文件存储输出接口 | Xxx.csv | Csv文件 |

大数据特征处理模块接口数据定义如下表4-3所示：

表 4‑3 大数据特征处理模块接口定义

|  |  |  |
| --- | --- | --- |
| 接口描述 | 数据形式 | 数据类型 |
| 题目信息输入接口 | Problem.csv | csv文件 |
| 用户信息输入接口 | User.csv | csv文件 |
| 提交信息输入接口 | Submission.csv | csv文件 |
| 日志信息输入接口 | Log.csv | csv文件 |
| 数据处理输出接口 | Uid:1, acNum:10, tag: dp | Spark DataFrame |
| 特征输出接口 | Vecotr [0.33,0.89…] | Embedding向量 |
| 数据存储接口 | Key:value | Redis数据 |

推荐模块接口数据定义如下表4-4所示：

表 4‑4 推荐模块接口数据定义

|  |  |  |
| --- | --- | --- |
| 接口描述 | 数据形式 | 数据类型 |
| Embedding向量输入 | Vecotr [0.33,0.89…] | Embedding向量 |
| 模型文件输出 | SavedModel | TensorFlow Model |
| 推荐服务模型导入 | SavedModel | TensorFlow Model |
| 在线推荐服务 | POST、GET | RESTful API |

系统业务模块接口数据定义如下表4-5所示：

表 4‑5 系统业务模块接口数据定义

|  |  |  |
| --- | --- | --- |
| 接口描述 | 数据形式 | 数据类型 |
| 用户注册与登录 | Email，Password | String |
| 组队推荐 | [User1, User2, User3 …] | List |
| 数据可视化 | AvgScore.png … | Image |
| 训练计划 | [TODO1, TODO2 …] | List |

各模块间的调用关系如下图4-3所示：

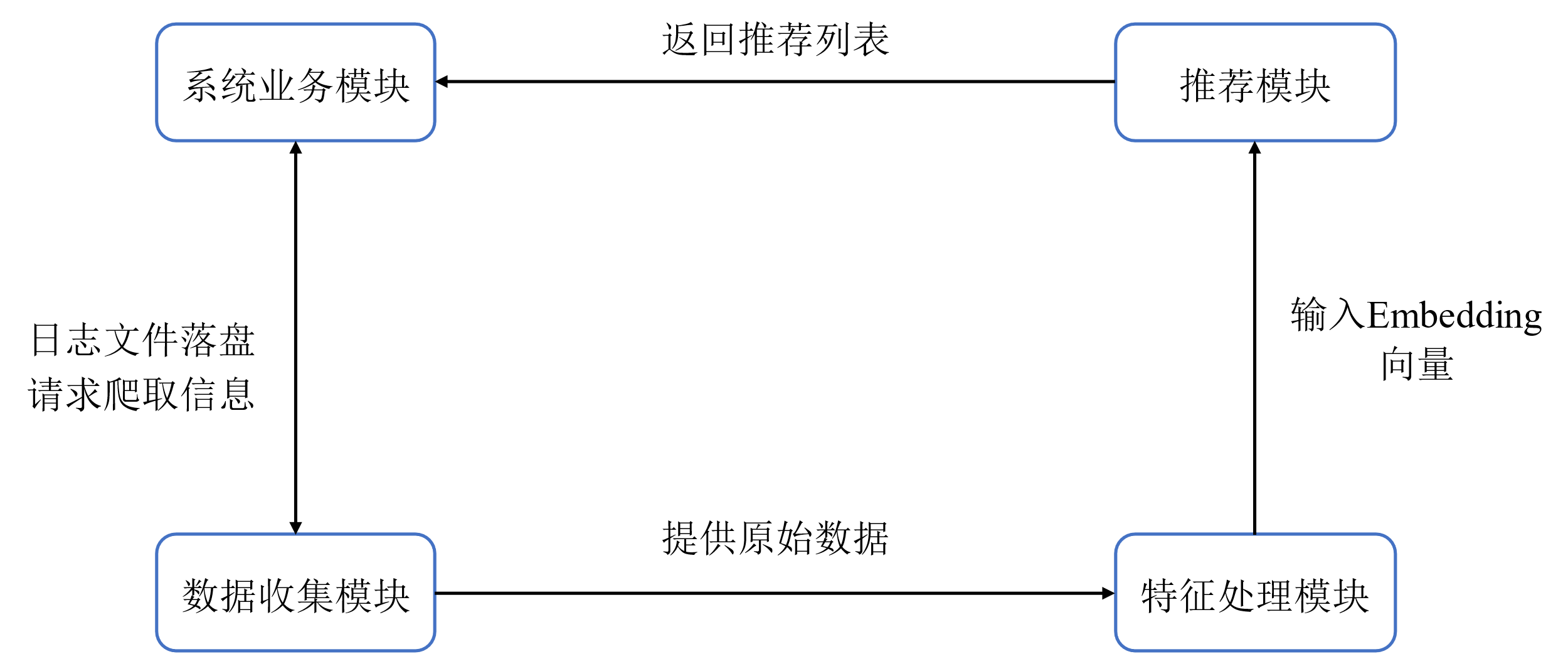


图 4‑3 模块间调用关系

### 技术路线选择

**1.** 系统业务编程语言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。本系统选择其作为系统业务和数据收集的原因有如下三点。

（1）性能优势：简单是Go语言的核心，它可以快速编译与运行并且快速上手开发。实测中Go拥有着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，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。

**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Python爬虫框架Scrapy，而是采用了Go语言的Colly爬虫框架。下表4-6展示了两者之间的差异：

表 4‑6 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 |

**3.** 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4-7展示了Zap与其他Go语言日志库的性能差距：

表 4‑7 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.** 分布式计算框架Spark

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4-8列出了Spark各个模块的功能：

表 4‑8 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。 |

## 系统详细设计

在系统详细设计阶段，主要对业务的实现流程、数据库的物理结构或数据的具体格式、核心功能实现等方面来对各个模块进行详细描述。

### 数据收集模块详细设计

**1.** 业务实现流程

离线收集的数据主要来源于对用户codeforces网站上的历史提交记录进行爬取，其流程如下图4-4所示：

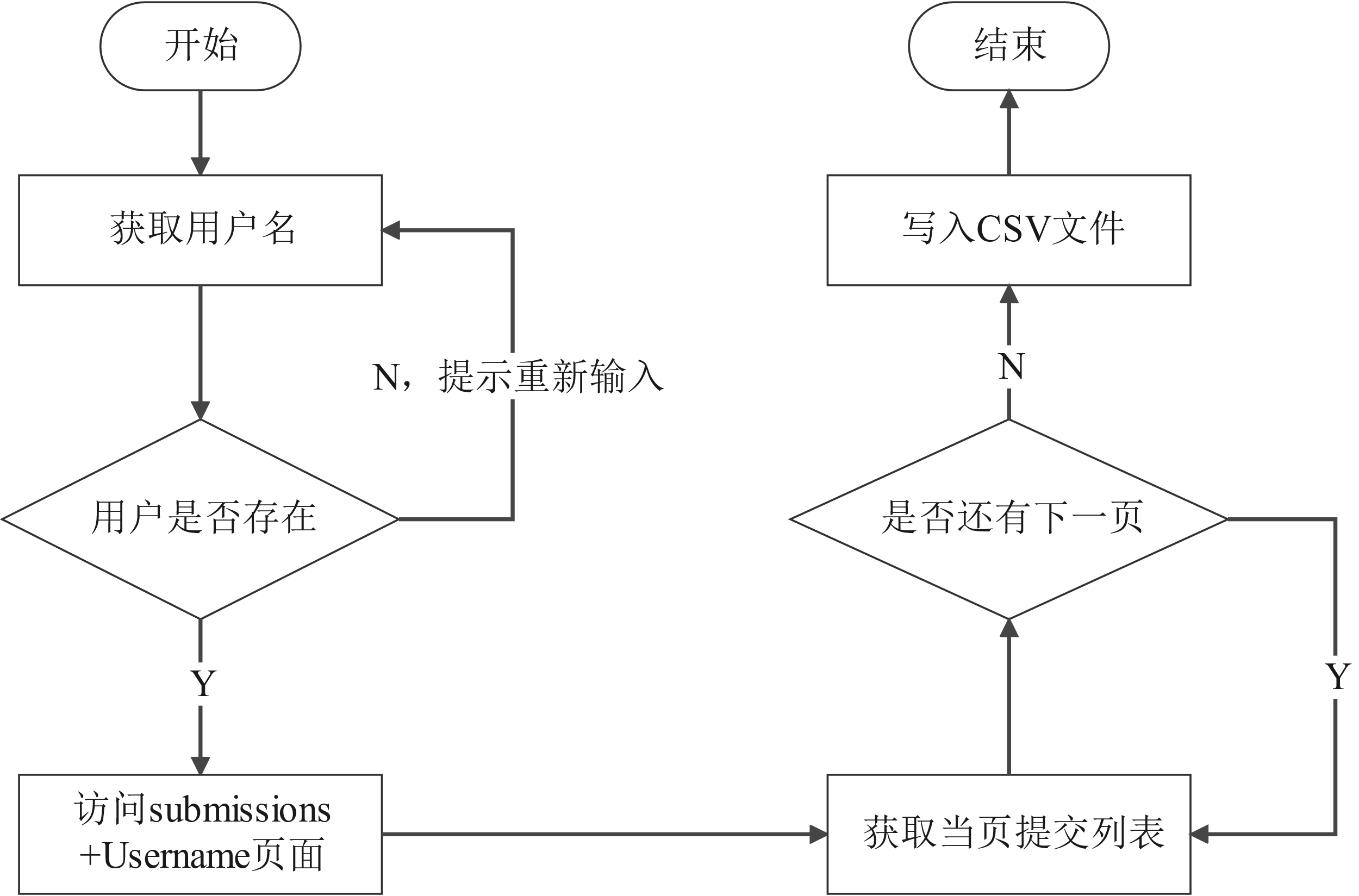


图 4‑4 用户提交历史记录获取流程图

获取用户信息与题目信息的流程如下图4-5所示：

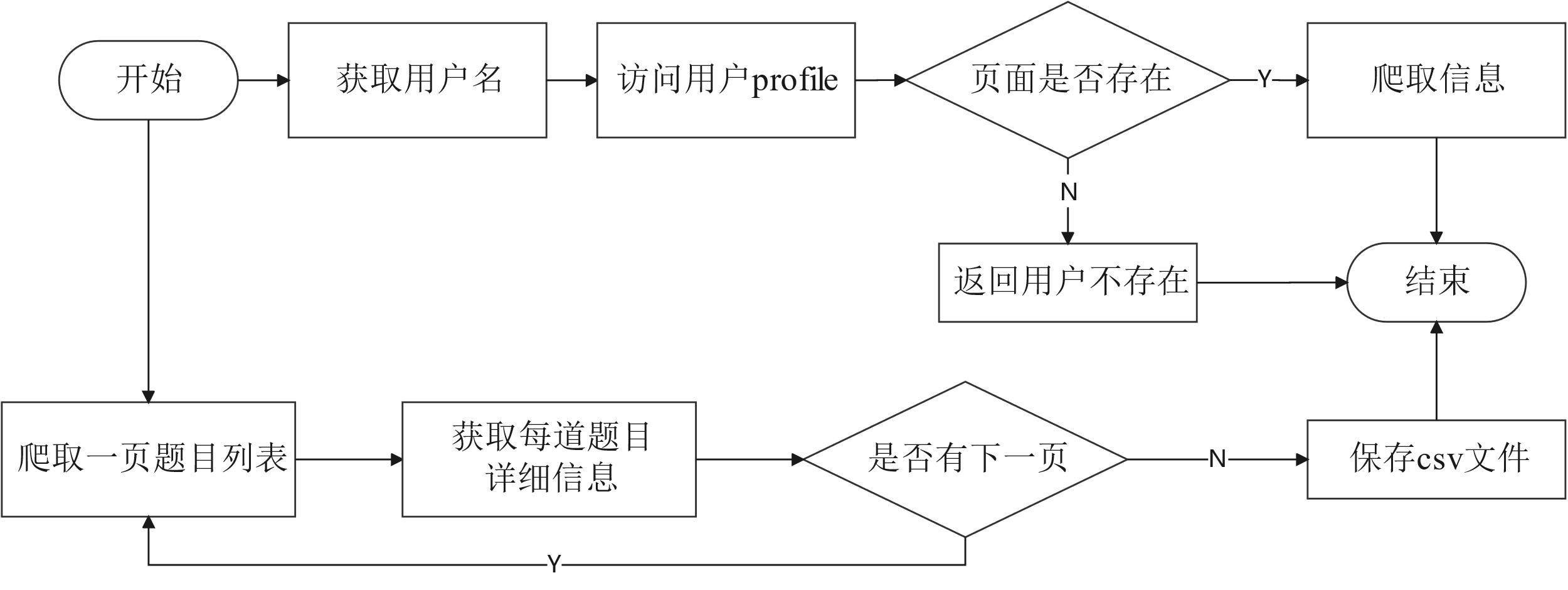


图 4‑5 获取用户和题目信息流程图

**2.** 爬取数据格式

爬取下来的数据分为三个csv文件存取进HDFS中，下面分别对其字段名和保存值得类型进行描述。

Problems.csv数据构成如下表4-9所示：

表 4‑9 Problems.csv数据构成

|  |  |  |
| --- | --- | --- |
| 字段名 | 存储类型 | 示例 |
| Pid | String | 1517H |
| Title | String | Fly Around the World |
| Score | Int | 3500 |
| Tags | String (split with #) | dp#geometry |

其中Pid为题目的编号，1517表示codefoces比赛的场次，H表示该题是当场比赛编号为H的题目；Title为题目名称；Score为该题目的难度分数；Tags为题目的分类标签，以“#”进行分隔。

Users.csv数据构成如下表4-10所示：

表 4‑10 User.csv数据构成

|  |  |  |
| --- | --- | --- |
| 字段名 | 存储类型 | 示例 |
| Uid | Int | 3 |
| UserRank | Int | 5 |
| UserName | String | tourist |
| ContestNum | Int | 197 |
| UserScore | Int | 3538 |

其中Uid为第一次爬取用户列表的固定编号；UserRank为每次爬取更新的用户排名；UserName为用户昵称；ContestNum为用户参加比赛场数；UserScore为每次比赛后用户的分数，其值的大小表示用户应有能力做出对应分值的题目。

Submissions.csv数据结构如下表4-11所示：

表 4‑11 Submissions.csv数据结构

|  |  |  |
| --- | --- | --- |
| 字段名 | 存储类型 | 示例 |
| Pid | String | 1517H |
| Uid | Int | 1 |
| SubTime | Timestamp | 1620643320 |
| Verdict | String | Accepted |

Uid和Pid分别表示上述两表的用户编号和题目编号；SubTime是一个时间戳类型表示从Jan/02/2006 15:04 (MST)到现在所经过的秒数；Verdict表示该用户本次的提交状态，Accepted表示题目通过，其他信息如Wrong Answer等表示提交错误。

**3.** 核心功能实现

go-colly框架提供给我们操作的是一个名为\*colly.Collector的结构体指针，为了避免每一种爬虫都需要重复设置参数，将其封装为GetCollector函数提供调用，其参数设置与含义如下表4-12所示：

表 4‑12 colly. Collector参数设置与含义

|  |  |  |
| --- | --- | --- |
| 参数 | 值 | 含义 |
| AllowedDomains | []string{"codeforces.com"} | 允许爬取网站列表 |
| Async | True | 是否开启并发爬取 |
| RoundRobinProxySwitcher | http://127.0.0.1:7890... | 循环代理池 |
| UserAgent | Chrome/86.0.4240.75… | 设置请求头 |
| RandomDelay | 100 \* time.Millisecond | 设置随机延迟 |
| Parallelism | 30 | 设置并发数 |

在设置完Collector参数之后，我们可以通过重写Go-Colly框架的钩子函数来实现爬取。其整个框架的生命周期如下图4-6所示：

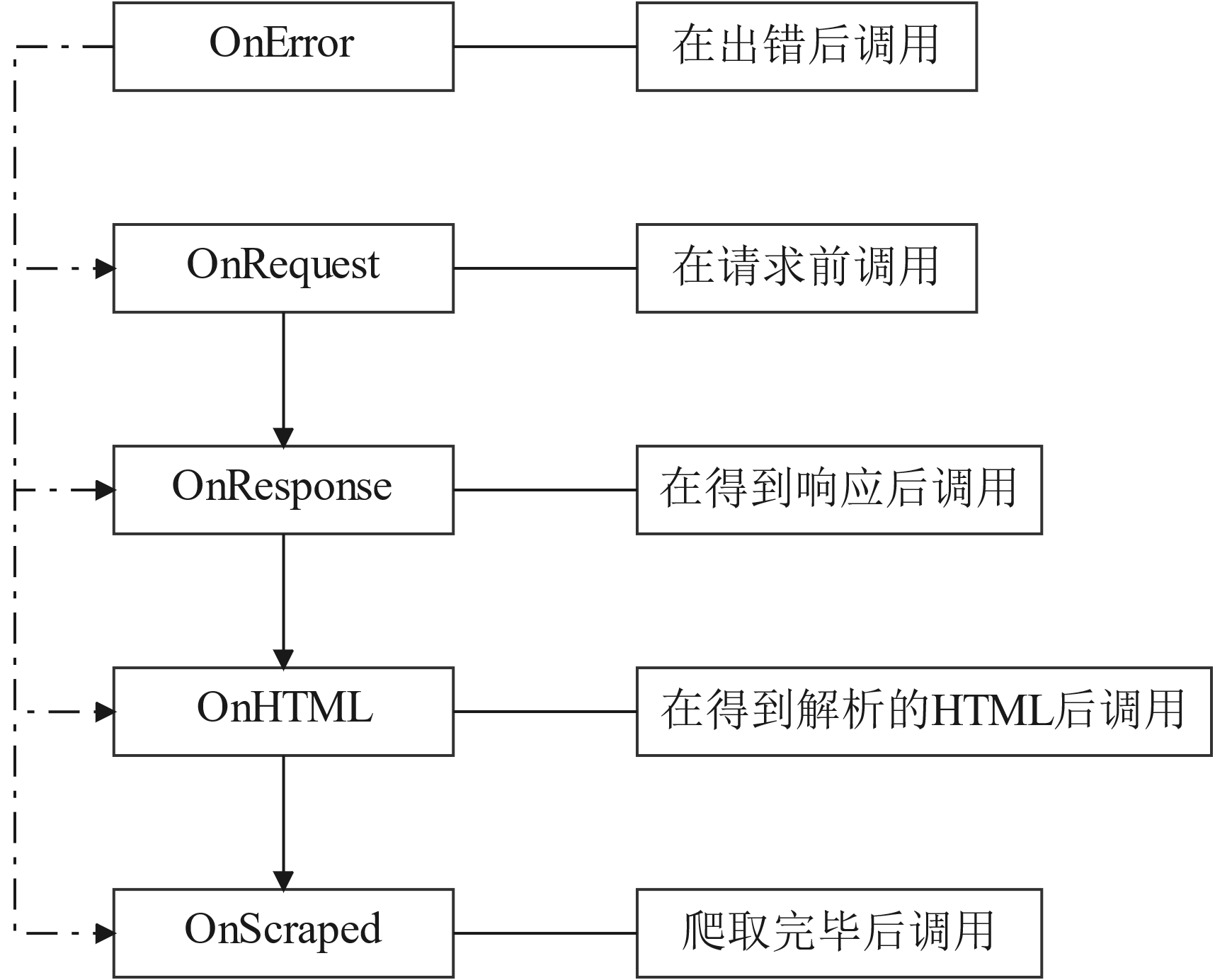


图 4‑6 gocolly框架生命周期

其中OnRequest函数可以让我们在请求前输出请求页面的url和设置随机的Headers来防止反爬虫机制对单一请求头的限制。OnError函数可以在爬取出现问题的时候抛出具体的问题类型，并对其进行相应的错误处理（如果不进行Panic操作的话程序不会退出）。OnHTML是进行爬取最关键的部分，它负责进行页面的解析，一般是通过goquery来对CSS选择器进行筛选。如：c.OnHTML(".status-frame-datatable > tbody", func(e \* colly. HTMLElement)。该语句选择了一个表格的tbody部分，并返回了一个colly.HTMLElement类型的结构体，让我们可以对其DOM进行操作。

最后爬取到的数据为一个结构体数组，通过第三方/ gocarina/gocsv包可以将其保存为csv文件。保存文件的功能如下表4-13所示：

表 4‑13 gocsv函数功能

|  |  |  |
| --- | --- | --- |
| 函数名 | 参数 | 功能 |
| MarshalCSVWithoutHeaders | in interface{}, out CSVWriter | 保存数据至CSVWriter |
| MarshalFile | in interface{}, file \*os.File | 保存数据至文件 |
| UnmarshalFile | in \*os.File, out interface{} | 读取CSV文件至接口 |

最终将保存好的csv文件上传至HDFS，CLI命令为：

hadoop fs -mkdir -p /input/cfdata // 创建文件夹 -p 表示级联创建

hadoop fs -put /submissions\*.csv /input/cfdata //上传所有的用户历史记录

至此，离线爬取数据阶段全部结束，并保存在HDFS中以供Spark特征处理模块进行处理和计算。

### 大数据特征处理模块详细设计

**1.** 业务实现流程

本模块利用pandas和spark处理数据如下图4-7、4-8所示：

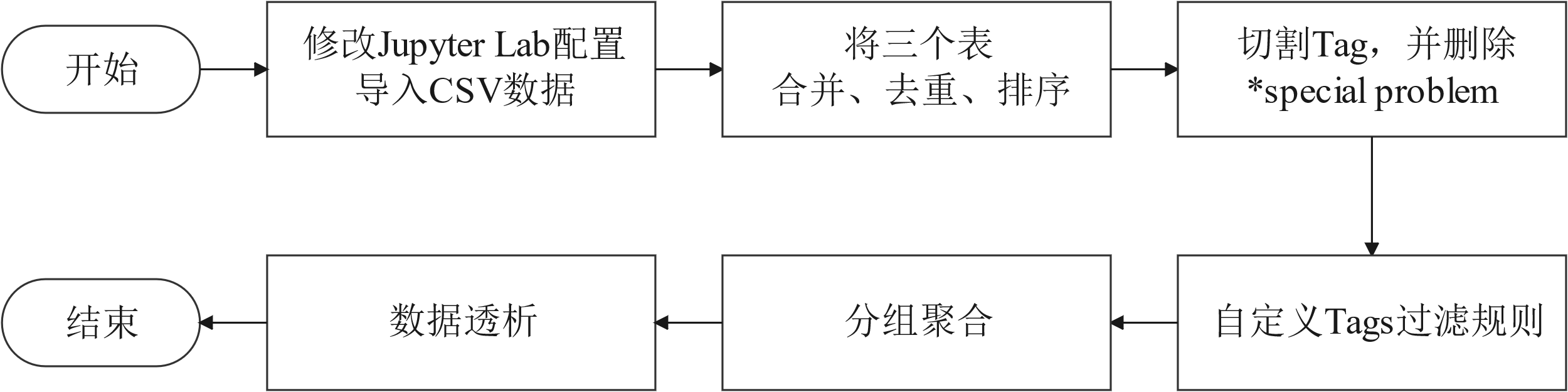


图 4‑7 pandas处理特征数据流程图

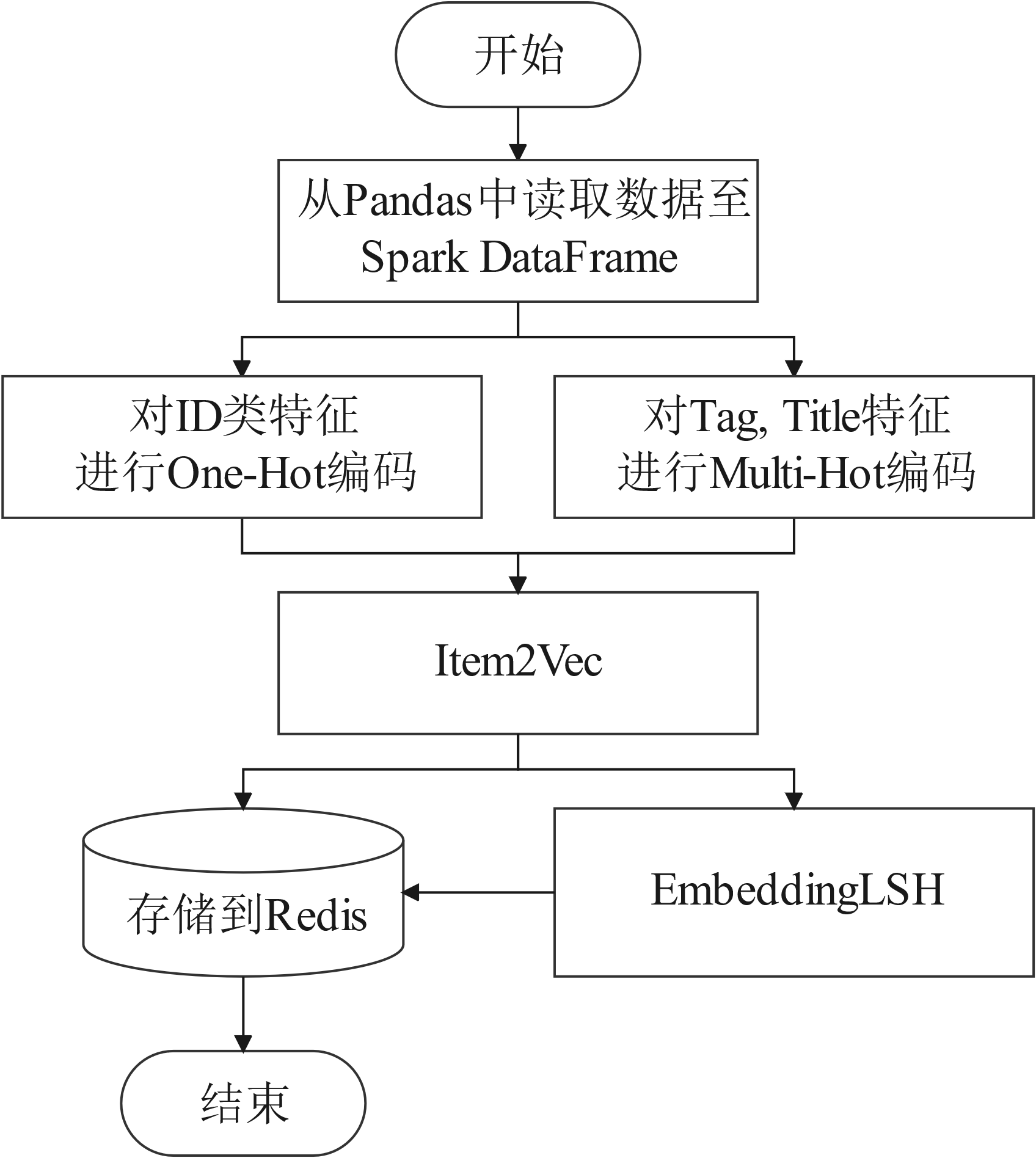


图 4‑8 spark处理数据特征流程图

首先从HDFS中加载csv文件到pandas的dataframe中，在pandas中将爬取得到的三张表进行合并，去除重复值、缺失值、异常值，并按照Uid进行排序。将Tags按照`#`号进行切割并自定义Tags过滤规则，将30种tag分成9个大类进行替换。然后开始分组聚合得到UserInfo、TagsInfo、SubmissionInfo等多个dataframe。最后对汇聚的表进行数据透析，得到每个用户在各个类别下的各项数值UTI和每个类别下最擅长的Top10个用户。

在对数据清洗完成之后，将pandas内的数据加载到spark中，对类别型的数据进行multi-hot编码，对ID类型数值进行one-hot编码。然后将处理好的特征进行Item2Vec训练得到Embedding向量，一份保存在Redis数据库以供线上训练使用，另一份直接进行局部敏感哈希来直接对题目进行召回，召回列表同样保存在redis中。

**2.** DataFrame数据格式

经过数据清洗和处理后形成若干个dataframe，其中UserInfo按照UserName聚合来表示每个用户的信息；UserTagsInfo和TagsUserInfo为两张数据透视表，分别表示每个用户在各个类别的表现情况和每个类别下各个用户的表现情况，它们都具有多个索引（表中加粗项）。

（1）UserInfo如下4-14表所示：

表 4‑14 UserInfo数据结构

|  |  |  |
| --- | --- | --- |
| 字段名 | 属性 | 含义 |
| Uid | Int | 用户ID |
| Username | String | 用户名 |
| Score\_avg | float | 用户通过题目平均分数 |
| Score\_max | Int | 用户通过题目最大分数 |
| Ac\_num | Int | 用户通过题目数量 |

（2）UserTagsInfo如下表4-15所示：

表 4‑15 UserTagsInfo数据结构

|  |  |  |
| --- | --- | --- |
| 字段名 | 属性 | 含义 |
| Uid（index） | Int | 用户ID |
| UserName（index） | String | 用户名 |
| Tags（index） | String | 类别名称 |
| ac\_num | Int | 该类别下某用户通过题目数量 |
| tags\_max | Int | 该类别下某用户通过最大分数 |
| tags\_avg | Float | 该类别下某用户通过平均分数 |
| tags\_mid | Int | 该类别下某用户分数的中位数 |
| tags\_qua | Int | 该类别下某用户分数的上四分位数 |
| tags\_std | Float | 该类别下某用户分数的标准差 |

（3）TagsUserInfo如下表4-16所示：

表 4‑16 TagsUserInfo数据结构

|  |  |  |
| --- | --- | --- |
| 字段名 | 属性 | 含义 |
| Tags（index） | String | 类别名称 |
| Uid（index） | Int | 用户ID |
| UserName（index） | String | 用户名 |
| ac\_num | Int | 该用户在某类别下通过的题目数量 |
| tags\_max | Int | 该用户在某类别下通过的最大分数 |
| tags\_avg | Float | 该用户在某类别下通过的平均分数 |
| tags\_mid | Int | 该用户在某类别下通过的中位数 |
| tags\_qua | Int | 该用户在某类别下通过的上四分位数 |
| tags\_std | Float | 该用户在某类别下通过分数的标准差 |

（4）UTI和TUI，它们是在UserTagsInfo和TagsUserInfo的基础上经过查询过滤而成，UTI对每个用户所擅长的类别进行了排序；TUI对每个类别下的用户取了Top10。它们的结构同上面两张表一致，不再单独列出。

**3.** 核心功能实现

（1）Tag替换

原数据有30多个tag分类，有很多分类重合并且不利于后续处理，将其使用python字典分成9个大类，分别为Basis（基础算法）、Math（数论）、Graph（图论）、Data Structures（数据结构）、DP（动态规划）、String（字符串处理）、Search（搜索）、Games（博弈论）以及Geometry（计算几何）。将原本以#分割的tags拆开，然后通过stack()操作将拆分后的列表堆叠，最后使用apply函数通过自定义规则进行替换并重新join进原来的列表。

（2）数据聚合与透视

数据聚合主要使用pandas的groupby语句，其参数含义如下表4-17所示：

DataFrame.groupby(by=None,axis=0,level=None,as\_index=True,sort=True, observed=False, dropna=True)

表 4‑17 pandas.groupby参数含义

|  |  |
| --- | --- |
| 参数 | 含义 |
| by | 用于确定GroupBy的值 |
| axis | 0表示按行分组，1表示按列分组 |
| level | 如果是多级索引，则设置分组的级别 |
| as\_index | 是否将分组列名作为输出的索引 |
| sort | 对组内的值进行排序 |
| observed | True显示 |
| dropna | 删除空值 |

数据透视主要使用了pandas的pivot\_table语句，其参数含义如下表4-18所示：

DataFrame.pivot\_table(values=None,index=None,columns=None,aggfunc='mean', fill\_value=None, margins=False, dropna=True)

表 4‑18 pivot\_table参数含义

|  |  |
| --- | --- |
| 参数 | 含义 |
| values | 用于分析的列 |
| index | 定义数据透视的索引 |
| columns | 指定表示数据元素类别的键值 |
| aggfunc | 将values列的值进行计算 |
| fill\_value | 指定空值填充 |
| margins | 是否添加所有的行和列 |
| dropna | 删除空值 |

UTI的生成主要是循环用户列表，并对UserTagsInfo进行query查询。查询语句为query('UserName == @username')，其中@表示了后面跟着的username为一个变量。最后通过sort\_values按照该用户在各个类别下的上四分位数、均值、标准差来进行排序。由于cf题目的分数差别较大，很多基础题目的分数为800分，而难题的分数为3500分，为了最大程度去除极端分数的影响而优先使用上四分位数排序。当上四分位数相同时采用均值排序，最后若均值也相同则采用标准差来衡量用户在该类别下的稳定情况。

TUI的生成与UTI基本相同，在查询时额外限制了参加排名的用户做题数量以及对排名列表取top 10。查询语句为：query('Tags == @tagsname & ac\_num >= 50').head(10)

（3）Item2Vec训练

使用pyspark训练Item2Vec的具体流程如下图4-9所示：

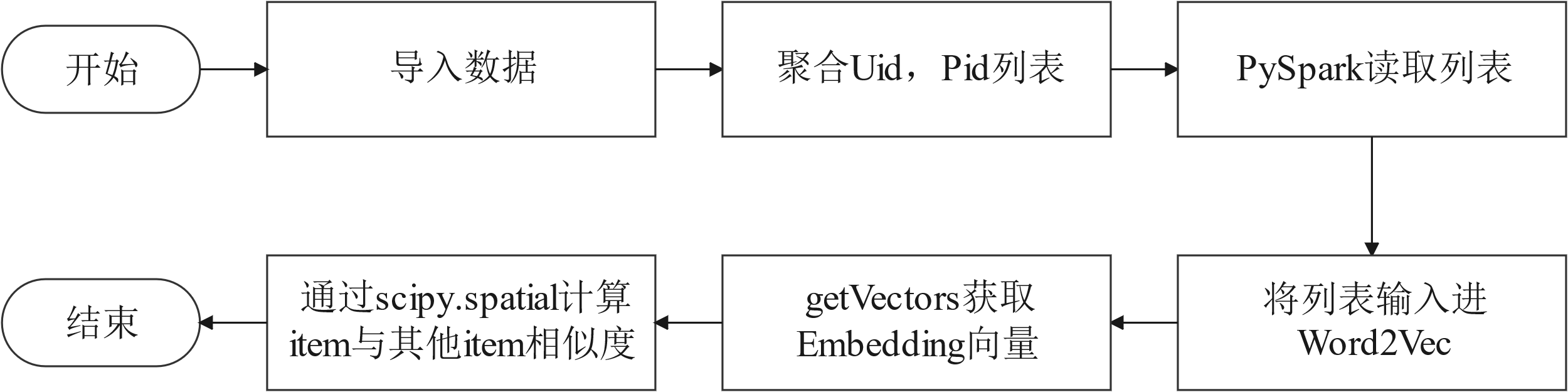


图 4‑9 pyspark训练Item2Vec流程

首先将用户提交记录submission.csv导入pandas中，对其按照Uid和Pid进行聚合，得到每个用户的提交列表。加载入pyspark中，使用spark.ml库中的word2vec对Item序列进行训练。得到model之后使用getVectors采取每道题目的Embedding向量。最终使用scipy对每道题目计算与其他题目的余弦相似度进行计算，经过排序后返回Top10列表。

其中，word2vec的相关参数设置如下表4-19所示：

表 4‑19 word2vec模型参数设置

|  |  |  |
| --- | --- | --- |
| **参数名** | **设置值 / 默认值** | **含义** |
| vectorSize | 25/100 | 生成Embedding向量维数 |
| learningRate | 0.025 | 学习率（学习步长） |
| numPartitions | 3 | 设置分区，取决于集群配置 |
| seed | Random | 随机种子 |
| minCount | 0 | 过滤最低词频 |
| windowSize | 5 | 滑动窗口大小 |

### 推荐模块详细设计

在数据特征处理模块通过用户的行为序列得到了Item\_Embedding，但是并没有获得User\_Embedding。通常的做法是将该用户所有做过题目的Item\_Embedding进行加权平均来得到User\_Embedding。但是这样会导致用户其他的特征无法得到充分利用，而深度学习恰恰可以弥补特征交叉不足。

**1.** 双塔模型结构

本文采用双塔模型，其结构如下图4-10所示：



图 4‑10 双塔模型结构

双塔模型的一大特点就是结构简单、易于应用。从NeaulCF模型得到得一大启发便是可以将User部分和Item部分分别处理，然后再通过点积运算或者MLP层获得二者的交叉信息。这样就可以轻易的对User和Item部分加入不同的特征，来达到更好的推荐效果。

本次设计的模型User塔部分输入为用户的编号Uid，为了防止生成的Embedding过大对其进行了索引处理，另一个输入为用户所有tags下的平均得分user\_tags\_score，并对其进行归一化处理。输入后分别对其进行Embedding操作，然后通过Concatenate操作进行特征拼接，形成一个大的User\_vector。通过两层全连接层（Dense）后生成一个8维的User\_Embedding向量。

Problem塔部分输入为题目的编号Problem\_idx，题目类型Tags\_idx和该题目的分数Score。同样经过Embedding后进行Concatenate特征拼接生成Problem\_vector向量。然后经过两个Dense层后生成Problem\_Embeding向量。

最终将User\_Embeding和Problem\_Embeding通过点积运算，得到预测的User\_TagsAvg，该值表示该用户在给定一组提交记录后，对某个标签的平均得分进行预估。损失函数使用平均平方误差（MSE），通过梯度下降来更新User\_embedding和Problem\_embedding的权重。

**2.** 输出值选取

对于推荐系统，模型本身的输出并不是我们所需要的，但是不合适的输出值会对embedding的生成造成很大的影响。本文初始选择的预测值是对题目的Score进行预测，最终生成的user\_embedding如下图4-11所示：

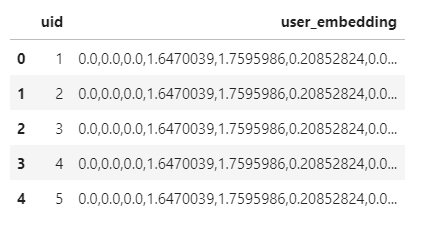


图 4‑11 使用Score进行预估的UserEmbeding

在模型收敛后，发现几乎每个User的Embeding向量都是一样的数值，初始认为是模型过拟合的问题，通过减少训练轮数和batch大小后embedding的值有所不同。但是通过仔细分析得出，若使用Score进行训练，那么每道题目的实际分值都是一样的，则用户对其分数进行预测导致每个用户的embedding向量必然相近。

最后本文选取了每个用户在该题目标签下的平均得分作为训练的目标，模型收敛后user\_embedding向量可以很大程度上反映出用户对于各种题目类别的能力。

**3.** 模型各参数说明

下表4-20展示了模型输入层参数定义：

表 4‑20 模型Input参数定义

|  |  |  |
| --- | --- | --- |
| 参数类型 | 参数值 | 含义 |
| keras.layers.Input | shape=(1,), name='uid' | 用户id |
| keras.layers.Input | shape=(1,), name='uscore' | 用户分数 |
| keras.layers.Input | shape=(1,), name='pid' | 题目id |
| keras.layers.Input | shape=(1,), name='tags' | 题目标签 |
| keras.layers.Input | shape=(1,), name=pscore | 题目分数 |

下表4-21展示了User塔的参数定义：

表 4‑21 User塔参数定义

|  |  |  |
| --- | --- | --- |
| 参数类型 | 参数值 | 含义 |
| layers.Embedding | (num\_users, 20)(uid) | 用户Embedding维度 |
| layers.Embedding | (num\_user\_score, 20)(uscore) | 分数Embedding维度 |
| tf.keras.layers.concatenate | name = 'user\_concatenate' | 用户向量特征拼接 |
| layers.Dense | 32, activation='relu' | 全连接层32维，激活函数为ReLu |
| layers.Dense | 8, activation='relu'  kernel\_regularizer='l2' | 全连接层8维，激活函数ReLu，使用L2正则化 |

下表4-22展示了Problem塔的参数定义：

表 4‑22 Problem塔参数定义

|  |  |  |
| --- | --- | --- |
| 参数类型 | 参数值 | 含义 |
| layers.Embedding | (num\_problems, 100)(pid) | 用户Embedding维度 |
| layers.Embedding | (num\_tags, 2)(tags) | 标签Embedding维度 |
| layers.Embedding | (num\_score, 20)(pscore) | 分数Embedding维度 |
| tf.keras.layers.concatenate | name = 'problem\_concatenate' | 题目向量特征拼接 |
| layers.Dense | 32, activation='relu' | 全连接层32维，激活函数为ReLu |
| layers.Dense | 8, activation='relu'  kernel\_regularizer='l2' | 全连接层8维，激活函数ReLu，使用L2正则化 |

上面三个表定义了整个模型的User塔和Problem塔，最后一个8维的全连接层就是所需要的Embedding向量。通过点积运算后使用一个1维的Dense层得到预测的分数，通过sigmoid函数将其归一化。得到预测输出。

定义损失函数平均平方误差loss=tf.keras.losses.MeanSquaredError()和优化器optimizer=keras.optimizers.RMSprop())

**4.** 模型训练

首先构建训练数据集，将数据集df使用sample采样百分之60，然后输入X选取"Uid\_idx","User\_Score", "Score","Pid\_idx", "Tags\_idx"列，输出y选取"User\_TagsAvg"列，其值为Tags\_User下九个类别分数的平均值。

指定TensorBorad的logs日志路径log\_dir = " /tf\_logs/logs/logs\_" +TIMESTAMP，以便可以在TensorBoard种查看损失率的变化情况以及模型的结构。

最后通过不断的尝试选取合适的训练值使模型收敛。其训练参数如下表4-23所示：

表 4‑23 model fit参数设置

|  |  |  |
| --- | --- | --- |
| 参数 | 参数值 | 含义 |
| x | fit\_x\_train | 模型输入 |
| y | y | 真实输出值 |
| batch\_size | 64 | 每次梯度更新的样本数 |
| epochs | 200 | 训练模型迭代次数 |
| verbose | 1 | 日志展示，1为显示进度条 |
| callbacks | tensorboard\_callback | 回调函数，用于TensorBoard |

**5.** 获取Embedding向量与模型保存

完成模型训练后，最终需要得到的是user和problem的embedding向量，这两个向量在双塔模型的位置是一个Dense层。TensorFlow支持传入部分参数来获取模型的某一层的值，model的输入如下图4-12所示：

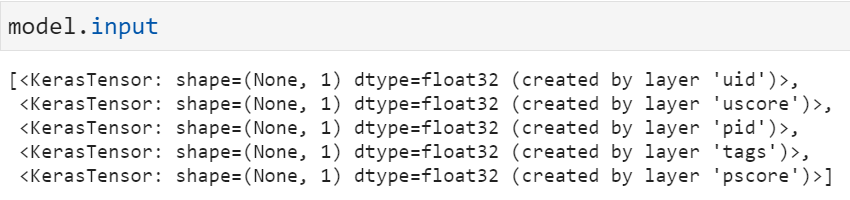


图 4‑12 双塔模型输入参数

从图中可以看出，如果想要获得User\_Embedding，那么需要传入input的前两个参数input[0]和input[1]，然后输出为model.get\_layer.output。将上文处理好的user列表传入既可得到User\_Embedding的值。然后使用pandas将其和uid重新组成dataframe保存，以便后续调用。Problem\_Embedding的获取方式同其一样，便不再赘述。

最后使用model.save('saved\_model/cfmodel')将训练好的模型进行保存，以便后续的训练和TensorFlow Serving的加载。

### 系统业务模块详细设计

系统业务模块的主要任务是将前面几个模块所得到的数据进行可视化显示，并通过User\_Embedding和Problem\_Embedding完成用户推荐和题目推荐。下图4-13为对本模块内各个功能间的关系：

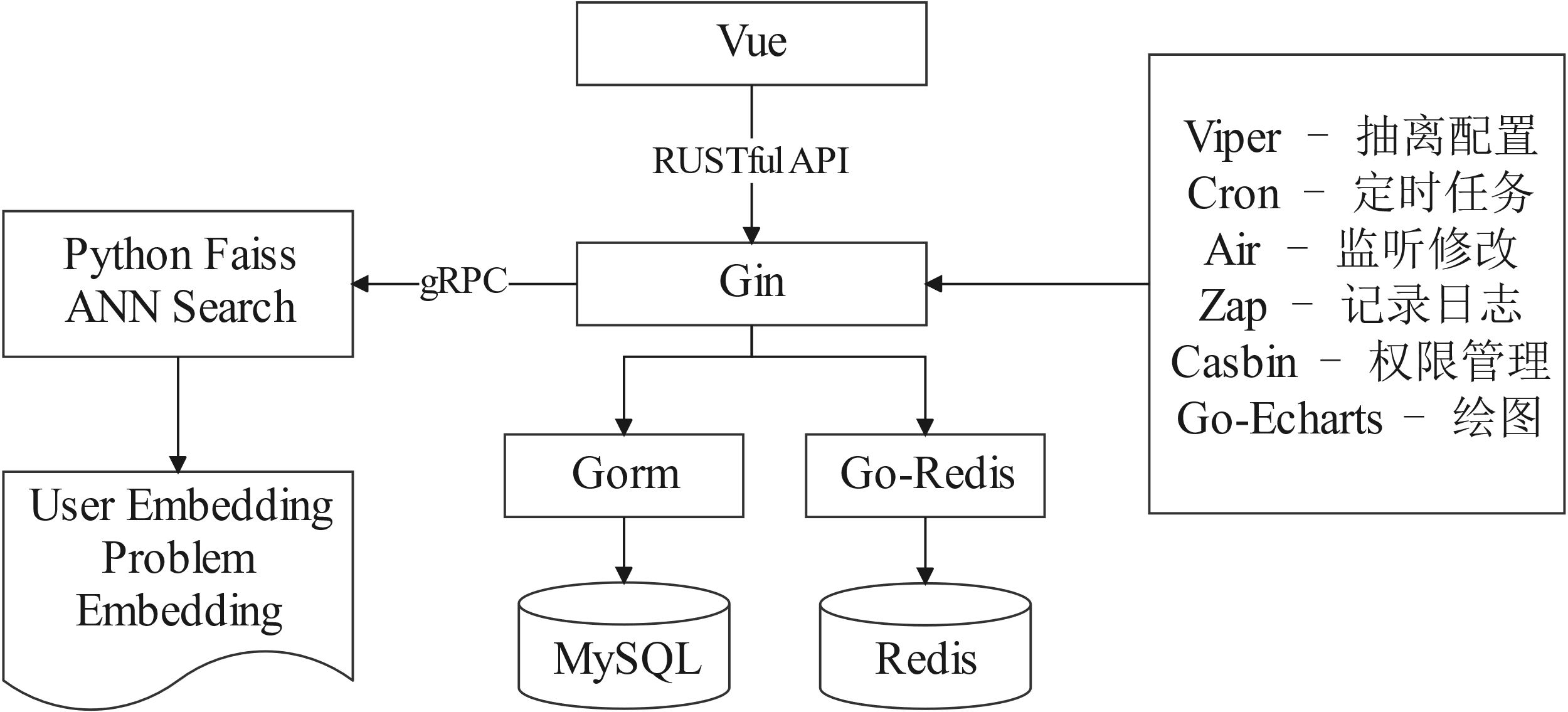


图 4‑13 系统业务模块功能间关系

**1.** 使用gRPC完成各模块间调用

由于各个模块间使用的语言各不相同，比如Web服务使用了Go语言，而用户的推荐列表和生成都需要python语言。基于微服务的思想本文使用gRPC来完成各个模块之间的调用。

gRPC的调用如下图4-14所示：

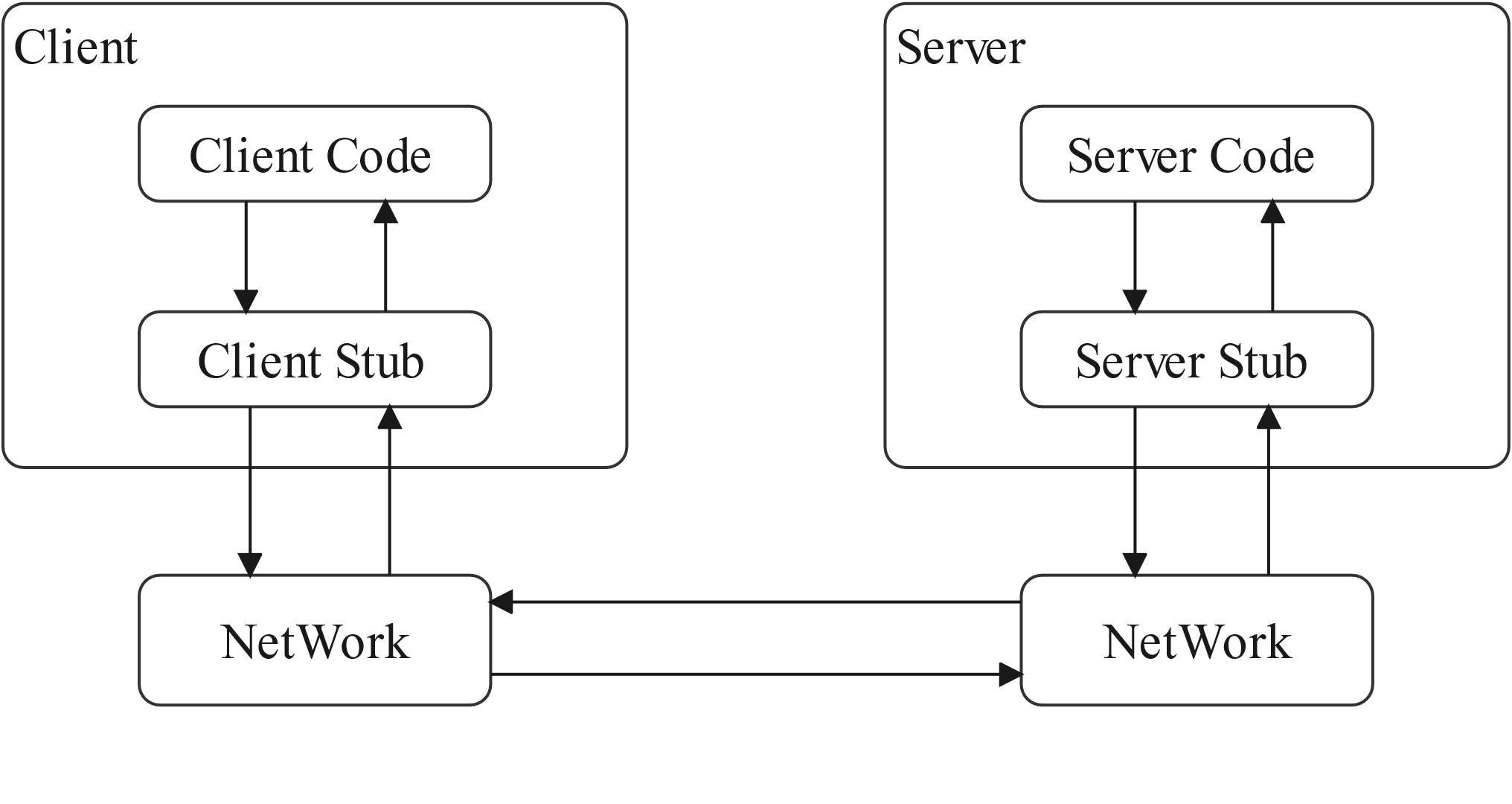


图 4‑14 gRPC调用结构

gRPC的思想是定义一个服务，指定其可以被远程调用的方法及其参数和返回类型。其默认使用 protocol buffers 作为接口定义语言，来描述服务接口和有效载荷消息结构。

在proto文件中定义的服务如下表4-24所示：

表 4‑24 proto文件中定义的服务

|  |  |  |
| --- | --- | --- |
| service | message | 含义 |
| PlotService | PlotRequest  PlotResponse | 请求绘制个人数据图表 |
| RecUserService | UserRequest  UserResponse | 请求返回队友推荐列表 |
| RecProblemService | ProblemRequest  ProblemResponse | 请求返回题目推荐列表 |
| CollyService | CollyRequest | 请求爬取数据 |

在定义完proto文件之后，可以使用go和python语言直接生产对应的服务端和客户端代码。Go语言中代码生成命令为：codegen.sh xxx.proto，python语言中代码生成命令为：protoc -I ../../protos --python\_out=. --grpc\_out =.—plugin = protoc-gen-grpc = `which grpc\_python\_plugin` xxx.proto。

**2.** 数据可视化

使用Vue-Echarts对用户各项信息可视化的流程如下图4-15所示：



图 4‑15 Vue-Echarts数据可视化流程

以可视化用户Benq在各个Tags下的平均得分为例，介绍可视化流程。首先通过浏览器访问Benq的个人空间，url=http://ip:port/profile/benq。加载页面时，Vue向后端Gin发送GET请求来获取所需展示的数据。请求的内容为{ uid:1 }。当Gin获得用户ID后通过查询数据库得到该用户在九个分类下的各项均值，然后返回给Vue界面。然后通过插件Vue-Echarts绘制雷达图进行可视化展示。

**3.** 获取推荐列表

使用Facebook开源的Faiss对Embedding向量进行ANN近邻搜索，生成对每个用户的推荐列表，其步骤如下图4-16所示：



图 4‑16 Faiss近邻搜索推荐列表流程图

使用Faiss获取Top10推荐列表的过程如下：

（1）加载User\_Embedding或Problem\_Embedding向量。

（2）将其索引转换为int64类型，Embedding值转换为np. np.float32。

（3）使用IndexFlatL2构建索引，添加待搜索的Embedding向量。

（4）取得目标Embedding，实现搜索得到ID列表和索引。

（5）将获取到的UserID列表通过gRPC返回给Go WebServer。

由于ACM竞赛是三人小队，所以还需要将已经组成团队的两名队员的Embedding进行相加平均，将平均后的Embedding向量加入Faiss索引中，得到团队最后一人的推荐列表。

## 本章小结

本章对系统进行了概要设计和详细设计。概要设计对系统的功能模块进行划分、选择技术路线并阐述了选取原因、对模块间接口数据进行了定义，明确了各模块间的调用关系。详细设计对系统四大模块的业务流程、数据库或数据表的结构以及核心功能的实现做了详细描述。

# 程序设计竞赛组队推荐系统实现

## 环境部署

推荐系统的大数据开发环境如下表5-1所示：

表 5‑1 集群环境部署

|  |  |  |  |
| --- | --- | --- | --- |
| 对应版本/角色 | 服务器集群 | | |
| 服务器1 | 服务器2 | 服务器3 |
| Hadoop集群 | hadoop-3.2.1.tar.gz | | |
| NodeManager  DataNode | SecondNodeManager  DataNode | DataNode |
| Spark集群 | spark-3.1.2-bin-hadoop3.2.tgz | | |
| Master  Worker1 | Woker2 | Woker3 |
| Go-Colly集群 | Go 1.16 | | |
| Worker1 | Worker2 | Worker3 |

### Hadoop集群部署

HDFS集群分布式部署结果如下图5-1所示：

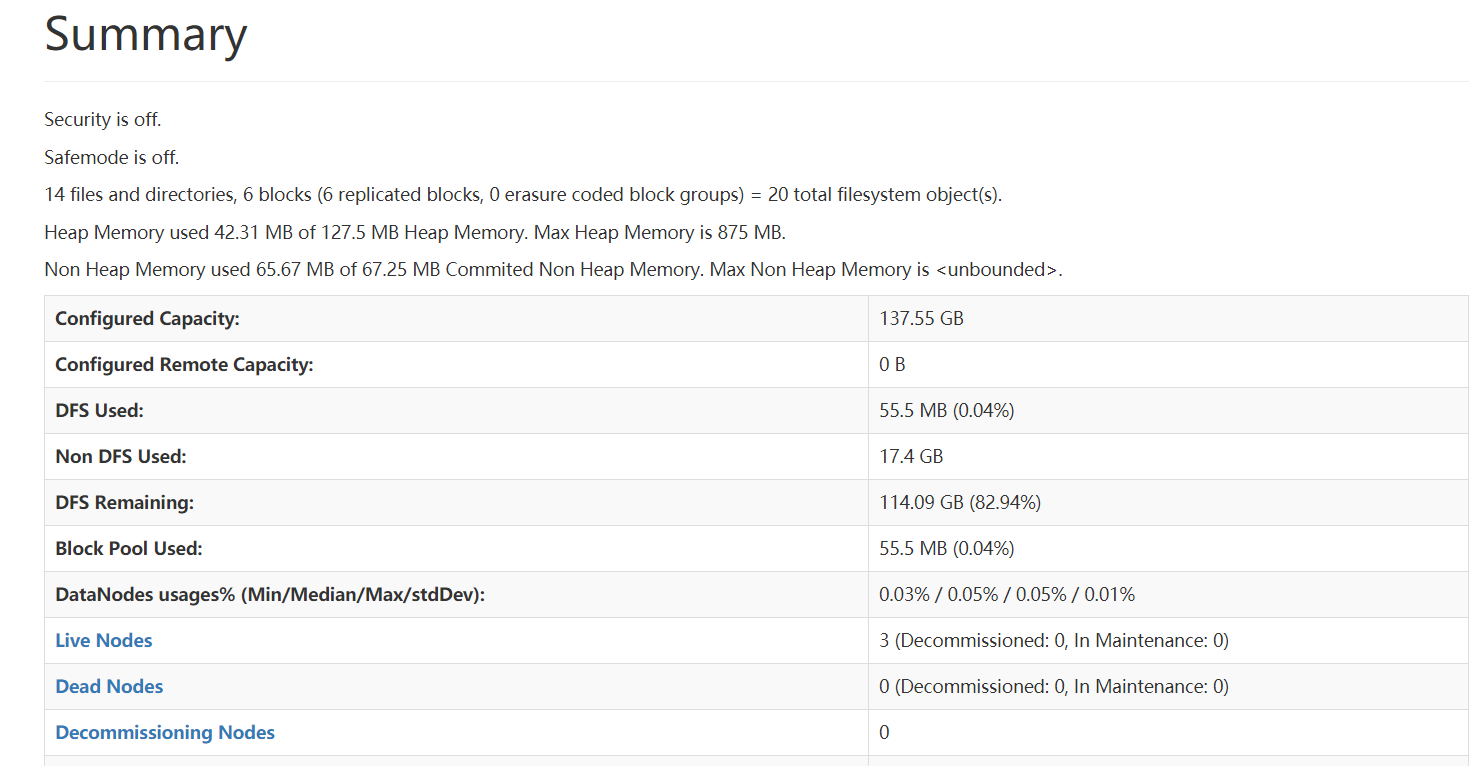


图 5‑1 HDFS集群分布式部署结果

Yarn集群部署结果如下图5-2所示：

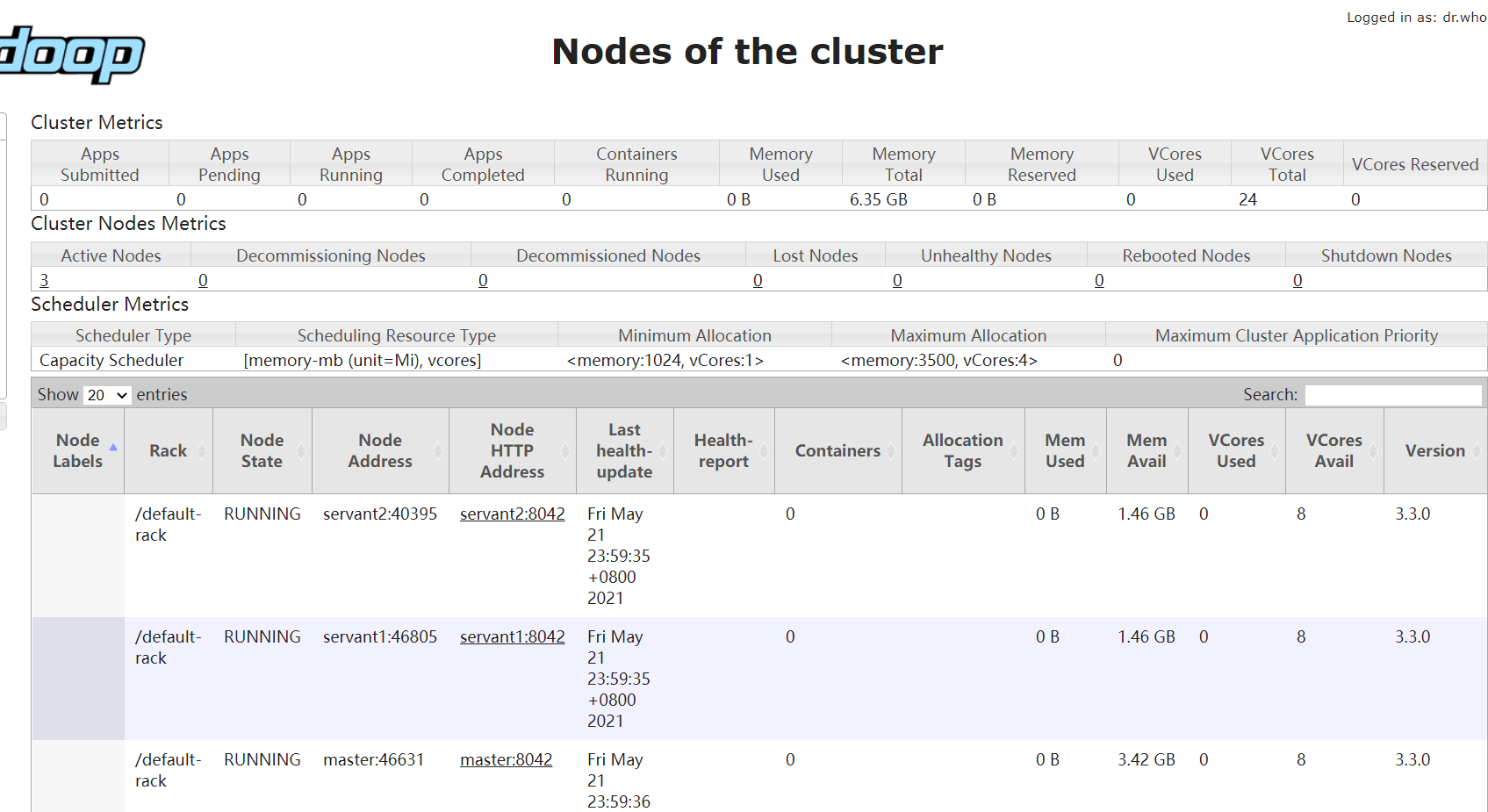


图 5‑2 Yarn集群分布式部署结果

### Spark集群部署

Spark集群分布式部署结果如下图5-3所示：

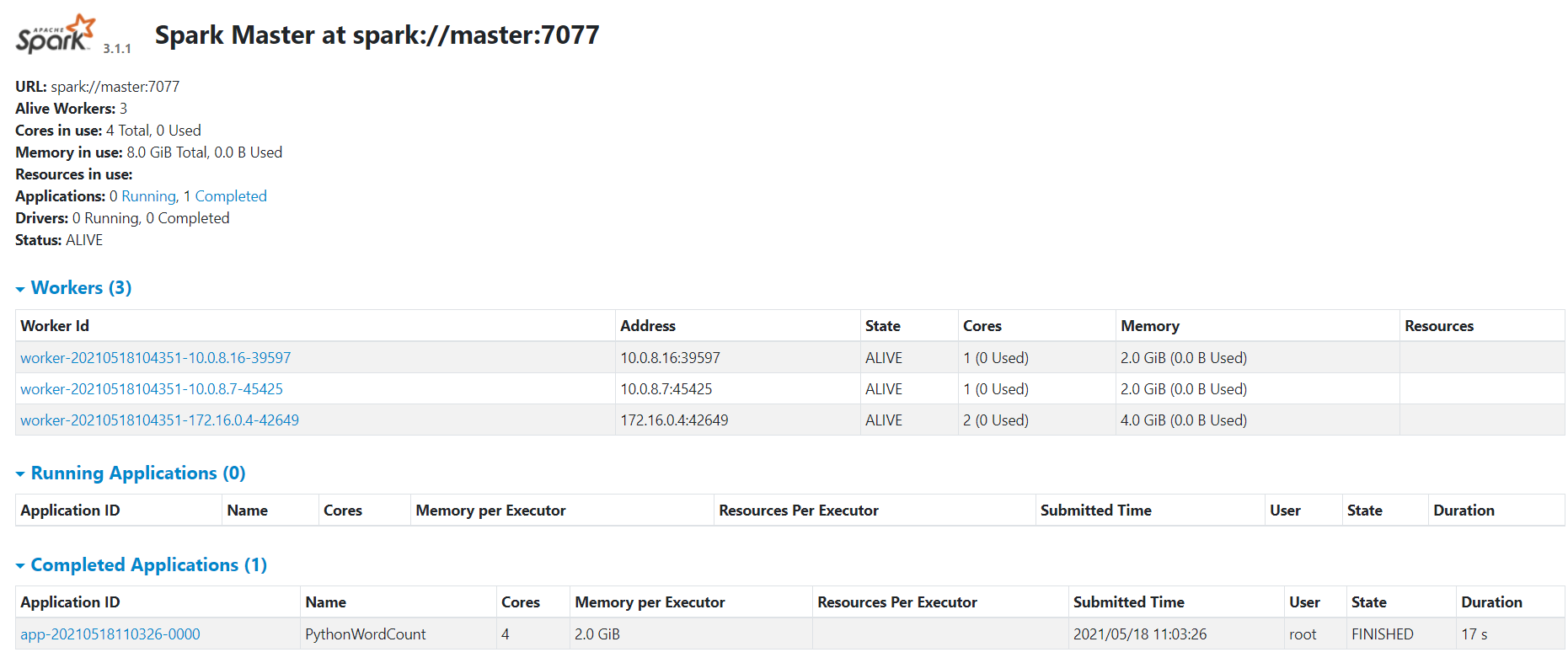


图 5‑3 Spark集群分布式部署结果

## 系统实现

### 数据收集模块实现

用户提交列表爬取结果如下图5-4所示：

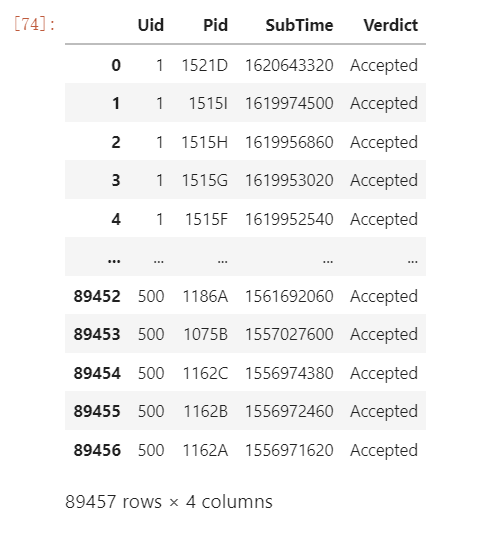


图 5‑4 用户提交列表爬取结果

从上图可以看出爬取的原始数据为89457条，将其保存为submission . csv文件。

用户爬取结果如下图5-5所示：



图 5‑5 用户信息爬取结果

6896条题目列表爬取结果如下图5-6所示：

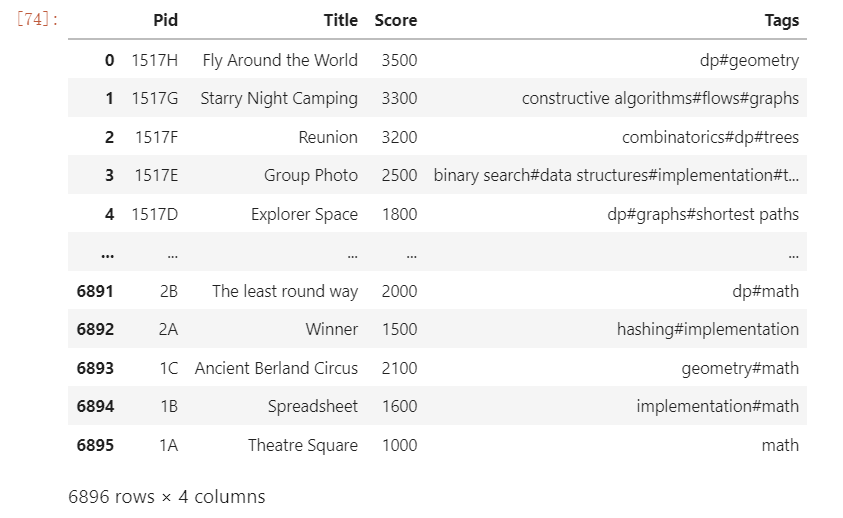


图 5‑6 题目列表爬取结果

### 特征处理模块实现

三个表经过合并、去重之后的结果如下图5-7所示：



图 5‑7 三表合并

可以看出经过去除无Tags提交和Score为0的提交记录之后，Submission的数据由89457下降到了76369条。

下图5-8展示了每个用户在九个大类别上的数值统计情况：分别为tag的均值、中位数、最大值、标准差、上四分位数以及在该tag下ac的题目数量。

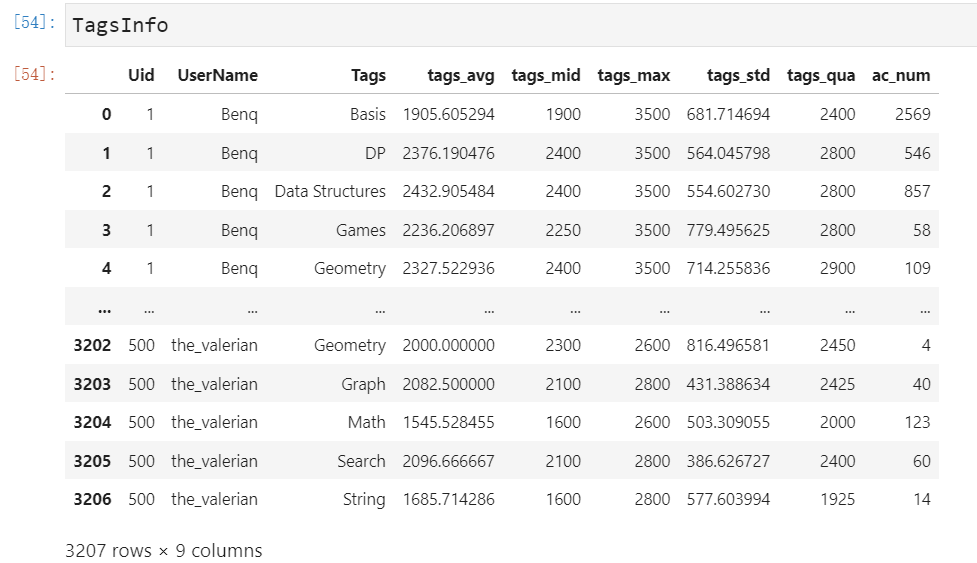


图 5‑8 每个用户在各个类别上的数值统计情况

下图5-9展示了经过数据透视后得到的每个用户在不同分类下的数据统计情况，其中每个用户的数值是一个单独的DataFrame：

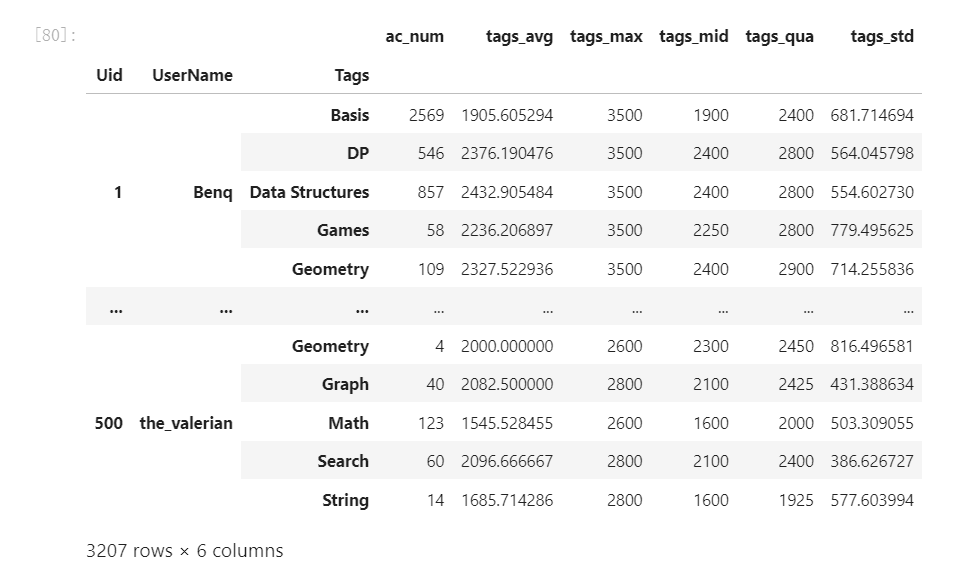


图 5‑9 用户Tags擅长情况数据透视表

下图5-10展示了经过Item2Vec后每个题目的Problem\_Embedding情况：

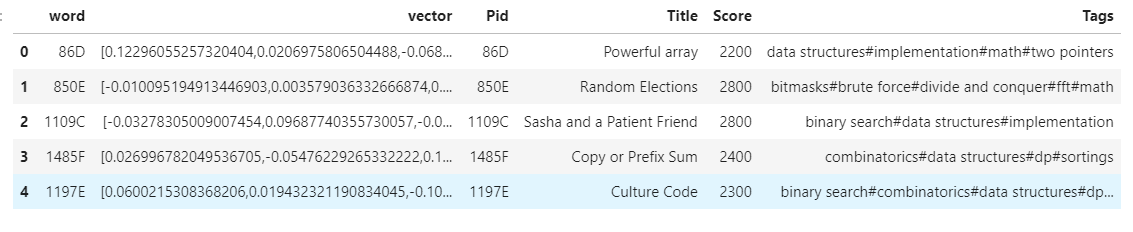


图 5‑10 Item2Vec Problem Embedding

图5-11展示了基于用户行为序列的Embedding召回情况：

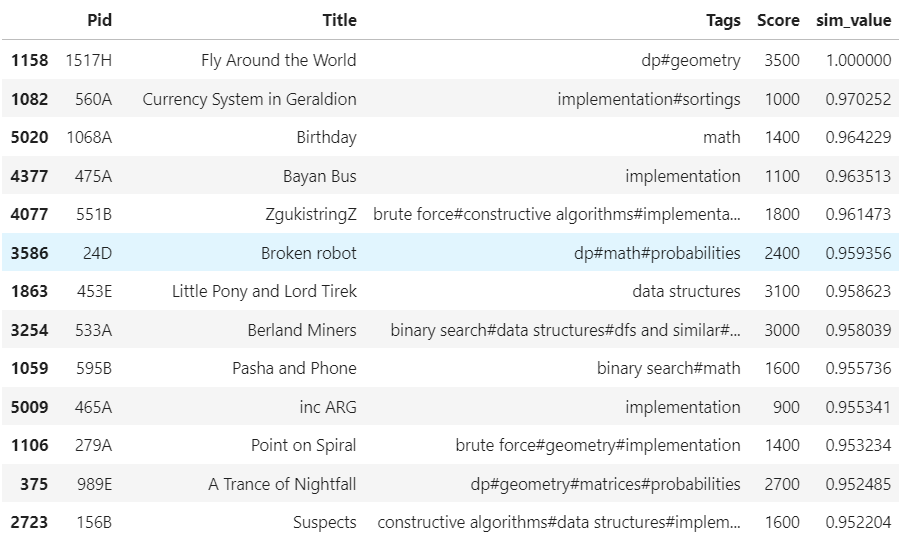


图 5‑11 基于用户行为序列的Embedding召回情况

从图中可以看出，基于用户行为序列的召回对于题目类别预测准确性不高，因为每场比赛选手们都会连续提交不同Tag的题目，但是增加了召回的多样性。

### 推荐模块实现

使用TensorFlow训练模型，其损失函数如下图5-13所示：

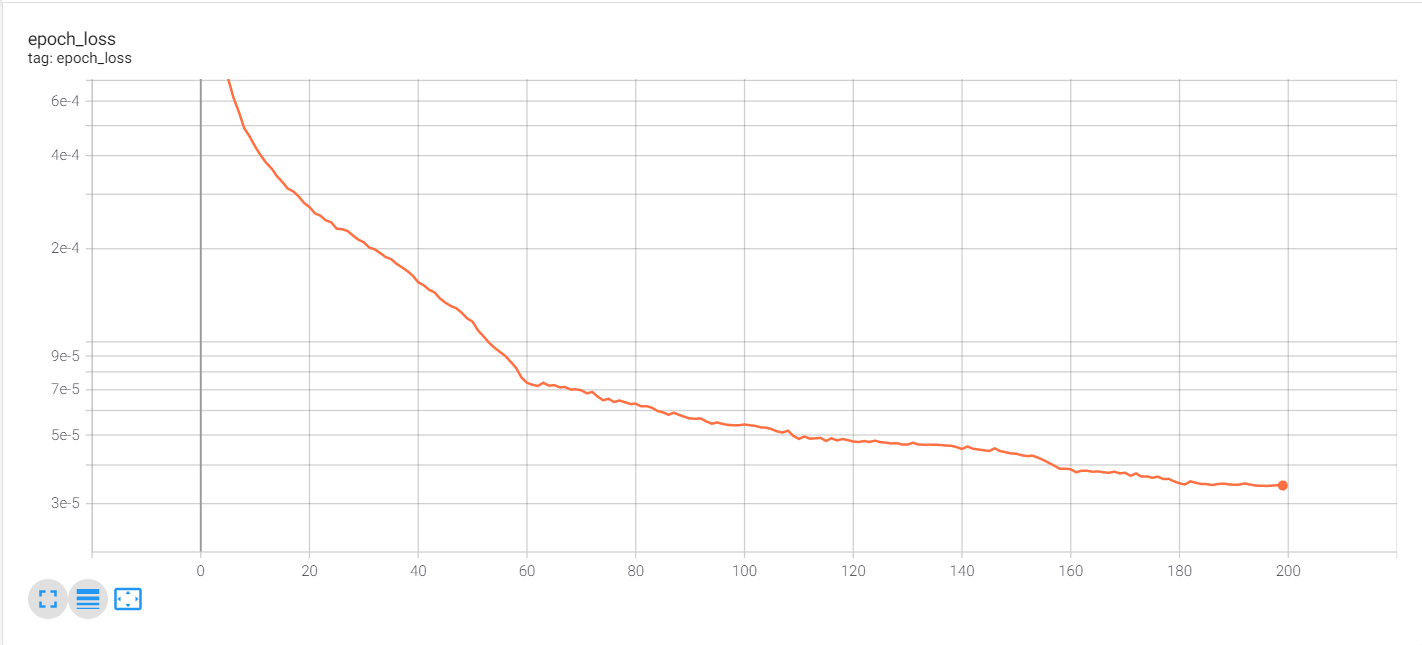


图 5‑12 Loss值变化图

从上图可以看出，模型在200轮作用基本收敛，loss值为3e-5到4e-5 之间。

图5-13、5-14展示了DSSM双塔模型训练出来的Problem\_Embedding和User\_Embedding：

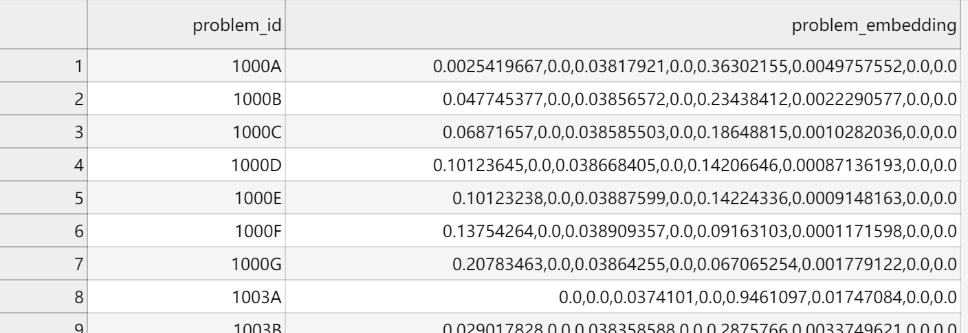


图 5‑13 DSSM Problem\_Embedding

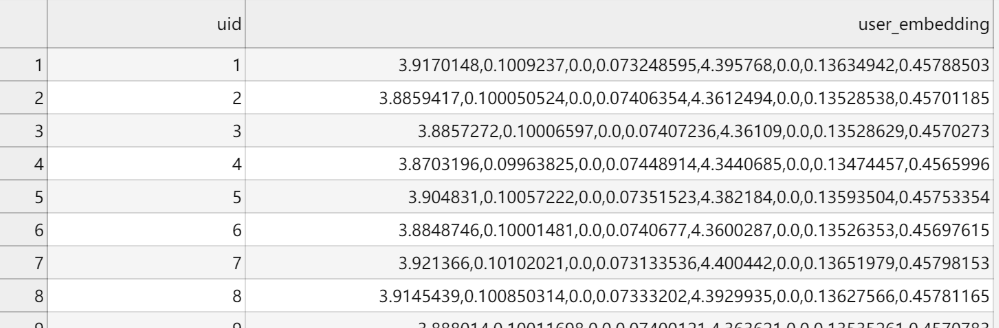


图 5‑14 DSSM User\_Embedding

### 系统业务模块实现

用户的登录界面如图5-15所示：

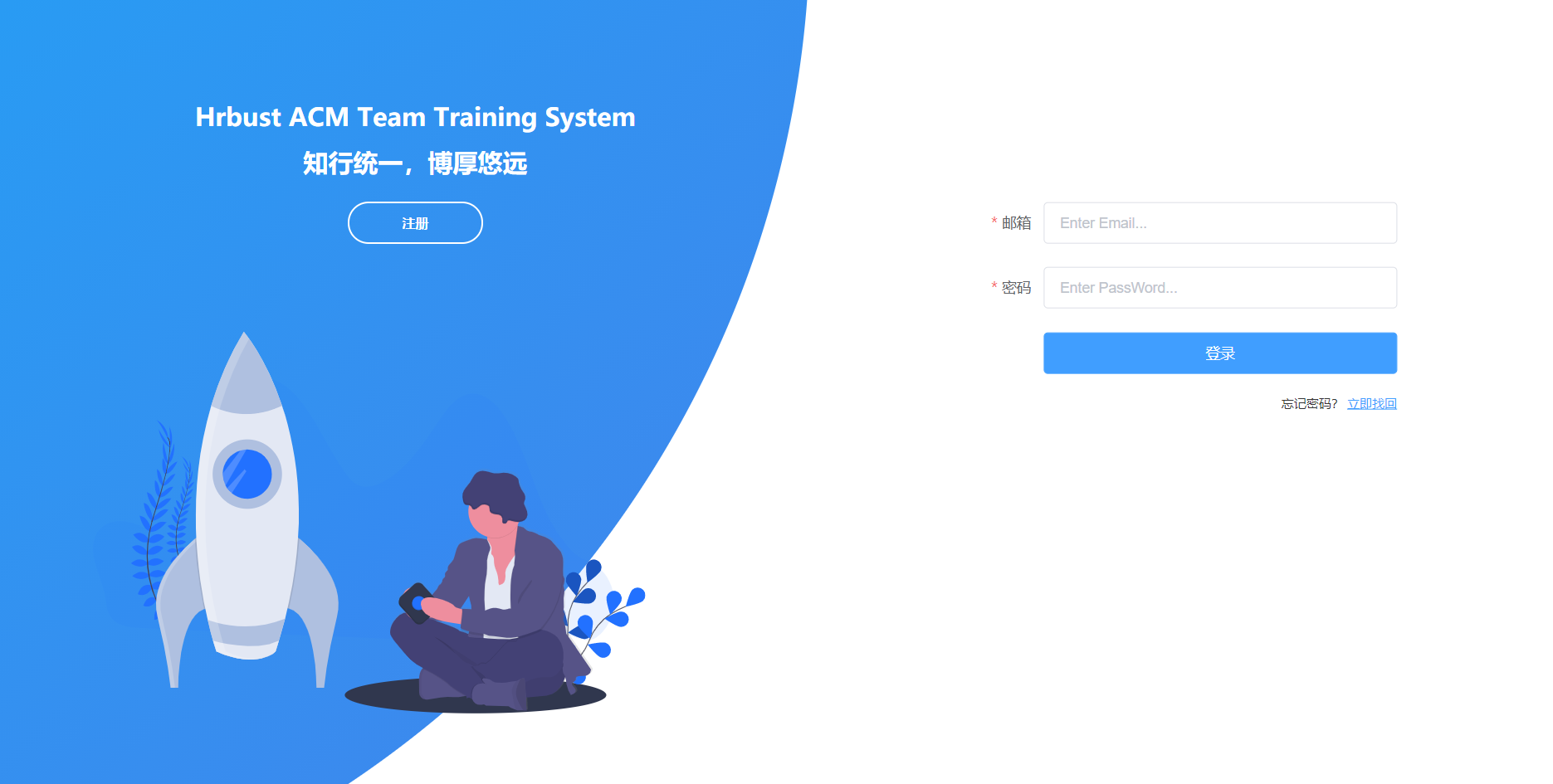


图 5‑15 登录页面实现结果

用户的注册界面如图5-16所示：

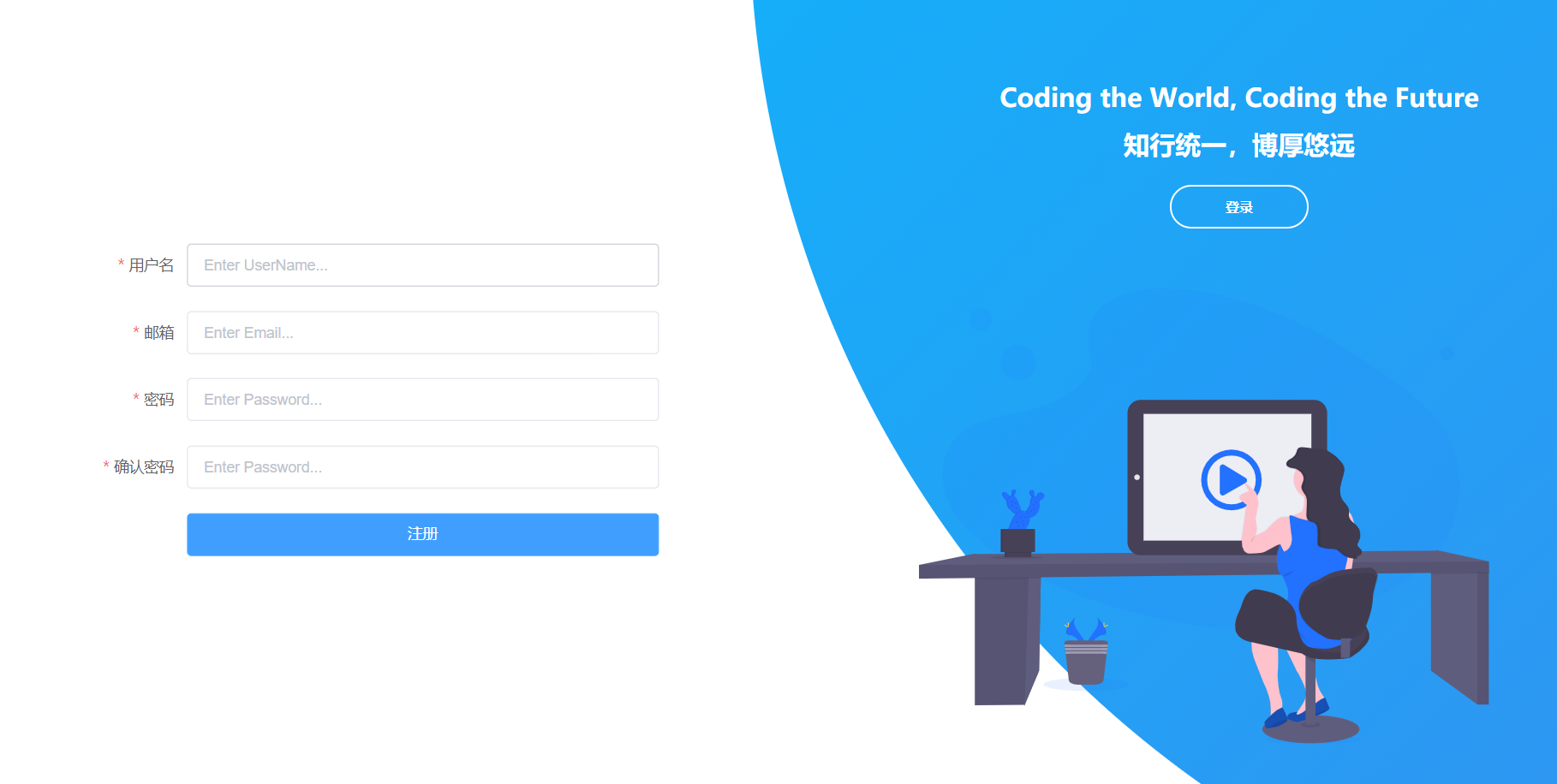


图 5‑16 注册页面实现结果

图5-17展示了在登录或注册时若输入错误信息的提示结果：

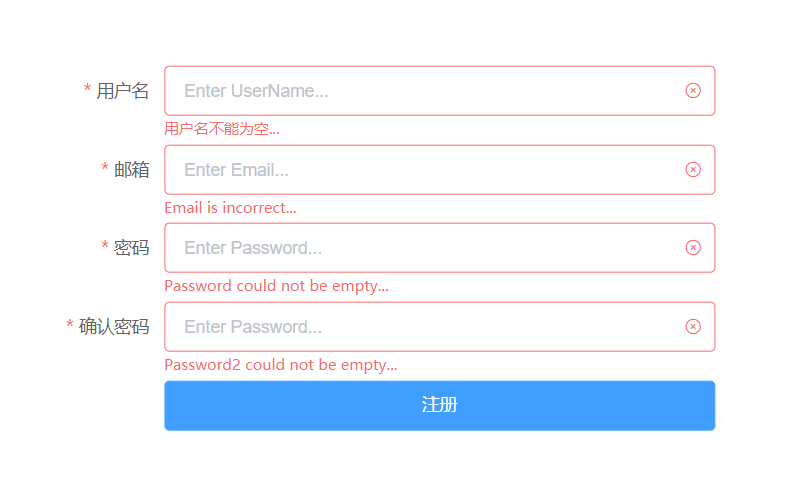


图 5‑17 错误信息提醒

图5-18展示了Wiki界面的样式结果：



图 5‑18 Wiki界面

## 本章小结

本章首先对由三台服务器组成的大数据处理平台进行了介绍，并展示其配置完成后的结果图。然后对系统数据爬取模块、特征处理模块、推荐模块以及前后端的系统业务模块进行了实现效果展示。

结论

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、个人数据可视化服务；为参赛团队提供了训练计划列表，团队知识覆盖可视化服务；为学校集训队提供了构建竞赛代码库服务。

本文进行的主要工作如下：

1. 在三台云服务器上构建了Hadoop与Spark集群，搭建了一套大数据分布式处理系统。

2. 使用Go语言的Colly框架对Codeforces网站进行爬取，共爬取了500名用户的89457条提交记录，以及网站内6896道题目的信息。

3. 使用pandas和pyspark对数据进行清洗，构建了数据透视表。并利用用户的提交记录生成了Problem\_Embedding向量，实现了基于用户行为序列的召回。

4. 通过对协同过滤算法、矩阵分解算法的实现原理与不足之处进行分析，并使用Embedding技术和DSSM双塔模型搭建并进行训练，得到经过特征交叉后的User\_Embeding和Problem\_Embedding向量。

5. 通过python faiss库对模型生成的Embedding向量进行ANN近邻搜索，得到用户和题目的推荐列表。

6. 通过vue-echarts库进行了数据可视化。

7. 使用gRPC技术实现了不同语言间的相互调用，形成了微服务架构。

8. 使用Vue和Gin框架实现了前后端分离的Web页面，将所有生成的数据进行展示。

本文存在的不足与对未来的展望如下：

1. 在召回策略上，没有通过多路召回来获取更加丰富的候选集。

2. 在生成基于用户行为序列的Problem\_Embedding时，只采用了Item2Vec技术，未能对有着更复杂结构的Graph2Vec技术进行尝试。

3. 在训练双塔模型时，用户塔部分只使用了2个特征，如果可以加入更多维度的特征，训练出来的User向量将会有更好的效果。

致谢

一谢师长：言传身教，诲人不倦：

首先要感谢我的论文指导老师高明老师。感谢您从论文选题开始一直到本文写作结束一直以来的信任、支持与指导。当我在无数次惶恐论文要写不完、写不好时，是您的激励可以让我沉下心来继续写作。感谢所有教导过我的老师们，是你们在这四年间一点一点的将计算机的基础知识变成我的计算机专业素养。最后，特别要感谢的是唐远新副院长，感谢您在我们17级计算机全体学生上倾注的心血。4年间，虽然您没有教我一门课，但是却教会了我做人的道理，偷奸耍滑只不过是自欺欺人，踏踏实实去做事才能够成功。再此衷心的感谢所有老师们四年间的辛勤付出。

二谢亲友：默默守护，同舟共济

感谢我的父母，这四年以来一直是你们的期盼与鼓励在伴我成长。在我失意之际的鼓励，骄燥之时的劝勉，都让我在人生的道路上越走越稳。感谢我445的室友们，能与你们一同生活和学习是我这四年最为幸运的事情，青山不改，友谊长存。

三谢母校：绿树长青，桃李满天

四年太短，短到来不及好好的感受和告别就要结束了。校园里的每一条道路上都有我的身影，这里承载了我整整四年的青春回忆。在这个六月，校园里的一草一木，一花一树，好像都在对我表达着依依不舍，即将远行，归期未卜。但无论如何，母校永远都是我们的家。

大学的时光到这里就要结束了，四年的时光仿佛一个轮回，首尾相应。新生入学伊始，ACM新生集训开启了我的大学生活；如今四年过去，依旧是在机房里敲下这份关于程序设计竞赛论文的最后一字。今当远离，再次向所有帮助、教导过我的老师、同学表示感谢。

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, 等. 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. 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.
5. Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[J]. arXiv preprint arXiv:1310.4546, 2013.
6. 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.
7. 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.
8. 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.
9. 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.
10. 景坤. 网约车竞赛平台的设计与实现[D]. 北京交通大学, 2019.
11. 胡开冉. 基于Hadoop/Spark的反向推荐算法研究[D]. 成都理工大学, 2018.
12. 李斌. 面向学科竞赛的组队平台研究[D]. 华中师范大学, 2020.
13. 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.
14. 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.
15. 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.
16. FANTOZZI P, LAURA L. Recommending Tasks in Online Judges using Autoencoder Neural Networks[J]. 2020.
17. 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.
18. 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.
19. 肖春芸, 贺樑, 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.

附录A

**英文原文**

**ITEM2VEC: NEURAL ITEM EMBEDDING FOR COLLABORATIVE FILTERING**

Abstract

Many Collaborative Filtering (CF) algorithms are itembased in the sense that they analyze item-item relations in order to produce item similarities. Recently, several works in the field of Natural Language Processing (NLP) suggested to learn a latent representation of words using neural embedding algorithms. Among them, the Skip-gram with Negative Sampling (SGNS), also known as word2vec, was shown to provide state-of-the-art results on various linguistics tasks. In this paper, we show that itembased CF can be cast in the same framework of neural word embedding. Inspired by SGNS, we describe a method we name item2vec for item-based CF that produces embedding for items in a latent space. The method is capable of inferring item-item relations even when user information is not available. We present experimental results that demonstrate the effectiveness of the item2vec method and show it is competitive with SVD.

**Index terms** – skip-gram, word2vec, neural word embedding, collaborative filtering, item similarity, recommender systems, market basket analysis, itemitem collaborative filtering, item recommendations.

1. INTRODUCTION AND RELATED WORK

Computing item similarities is a key building block in modern recommender systems. While many recommendation algorithms are focused on learning a low dimensional embedding of users and items simultaneously, computing item similarities is an end in itself. Item similarities are extensively used by online retailers for many different recommendation tasks. This paper deals with the overlooked task of learning item similarities by embedding items in a low dimensional space.

Item-based similarities are used by online retailers for recommendations based on a single item. For example, in the Windows 10 App Store, the details page of each app or game includes a list of other similar apps titled “People also like”. This list can be extended to a full page recommendation list of items similar to the original app as shown in Fig. 1. Similar recommendation lists which are based merely on similarities to a single item exist in most online stores e.g., Amazon, Netflix, Google Play, iTunes store and many others.

The single item recommendations are different than the more “traditional” user-to-item recommendations because they are usually shown in the context of an explicit user interest in a specific item and in the context of an explicit user intent to purchase. Therefore, single item recommendations based on item similarities often have higher Click-Through Rates (CTR) than user-to-item recommendations and consequently responsible for a larger share of sales or revenue.

Single item recommendations based on item similarities are used also for a variety of other recommendation tasks: In “candy rank” recommendations for similar items (usually of lower price) are suggested at the check-out page right before the payment. In “bundle” recommendations a set of several items are grouped and recommended together. Finally, item similarities are used in online stores for better exploration and discovery and improve the overall user experience. It is unlikely that a user-item CF method, that learns the connections between items implicitly by defining slack variables for users, would produce better item representations than a method that is optimized to learn the item relations directly.

Item similarities are also at the heart of item-based CF algorithms that aim at learning the representation directly from the item-item relations. There are several scenarios where item-based CF methods are desired: in a large scale dataset, when the number of users is significantly larger than the number of items, the computational complexity of methods that model items solely is significantly lower than methods that model both users and items simultaneously. For example, online music services may have hundreds of millions of enrolled users with just tens of thousands of artists (items).

In certain scenarios, the user-item relations are not available. For instance, a significant portion of today’s online shopping is done without an explicit user identification process. Instead, the available information is per session. Treating these sessions as “users” would be prohibitively expensive as well as less informative.

Recent progress in neural embedding methods for linguistic tasks have dramatically advanced state-of-the-art NLP capabilities. These methods attempt to map words and phrases to a low dimensional vector space that captures semantic relations between words. Specifically, Skip-gram with Negative Sampling (SGNS), known also as word2vec, set new records in various NLP tasks and its applications have been extended to other domains beyond NLP.

In this paper, we propose to apply SGNS to item-based CF. Motivated by its great success in other domains, we suggest that SGNS with minor modifications may capture the relations between different items in collaborative filtering datasets. To this end, we propose a modified version of SGNS named item2vec. We show that item2vec can induce a similarity measure that is competitive with an item-based CF using SVD, while leaving the comparison to other more complex methods to a future research.

The rest of the paper is organized as follows: Section 2 overviews the SGNS method. Section 3 describes how to apply SGNS to item-based CF. In Section 4, we describe the experimental setup and present qualitative and quantitative results.

2. SKIP-GRAM WITH NEGATIVE SAMPLING

Skip-gram with negative sampling (SGNS) is a neural word embedding method that was introduced by Mikolov et. al in [8]. The method aims at finding words representation that captures the relation between a word to its surrounding words in a sentence. In the rest of this section, we provide a brief overview of the SGNS method.

Given a sequence of words from a finite vocabulary , the Skip-gram objective aims at maximizing the following term:

where is the context window size (that may depend on ) and is the softmax function:

where and are latent vectors that correspond to the target and context representations for the word , respectively, and the parameter m is chosen empirically and according to the size of the dataset. Using Eq. (2) is impractical due to the computational complexity of , which is a linear function of the vocabulary size W that is usually in size of .

3. ITEM2VEC – SGNS FOR ITEM SIMILARITY

In the context of CF data, the items are given as user generated sets. Note that the information about the relation between a user and a set of items is not always available. For example, we might be given a dataset of orders that a store received, without the information about the user that made the order. In other words, there are scenarios where multiple sets of items might belong to the same user, but this information is not provided. In Section 4, we present experimental results that show that our method handles these scenarios as well.

We propose to apply SGNS to item-based CF. The application of SGNS to CF data is straightforward once we realize that a sequence of words is equivalent to a set or basket of items. Therefore, from now on, we will use the terms “word” and “item” interchangeably.

By moving from sequences to sets, the spatial / time information is lost. We choose to discard this information, since in this paper, we assume a static environment where items that share the same set are considered similar, no matter in what order / time they were generated by the user. This assumption may not hold in other scenarios, but we keep the treatment of these scenarios out of scope of this paper.

Since we ignore the spatial information, we treat each pair of items that share the same set as a positive example. This implies a window size that is determined from the set size. Specifically, for a given set of items, the objective from Eq. (1) is modified as follows:

Another option is to keep the objective in Eq. (1) as is, and shuffle each set of items during runtime. In our experiments we observed that both options perform the same.

The rest of the process remains identical to the method described in Section 2. We name the described method item2vec.

In this work, we used as the final representation for the i-th item and the affinity between a pair of items is computed by the cosine similarity. Other options are to use , the additive composition, or the concatenation . Note that the last two options sometimes produce superior representation.

附录B

**中文译文**

**ITEM2VEC：用于协作过滤的神经序列Embedding**

摘要

许多协同过滤（CF）算法是基于项目的，即它们分析项目与项目之间的关系，以产生项目的相似性。最近，自然语言处理（NLP）领域的一些工作建议使用神经嵌入算法来学习单词的潜在表示。其中，Skip-gram with Negative Sampling（SGNS），也被称为word2vec，被证明在各种语言学任务中提供了最先进的结果。在本文中，我们展示了基于项目的CF可以在同样的神经词嵌入框架下进行。受SGNS的启发，我们描述了一种我们命名为item2vec的方法，用于基于项目的CF，在一个潜在的空间中产生项目的嵌入。即使在没有用户信息的情况下，该方法也能够推断出项目与项目的关系。我们提出了实验结果，证明了item2vec方法的有效性，并表明它与SVD有竞争力。

索引词 - Skip-gram, word2vec, 神经词嵌入, 协同过滤, 项目相似性, 推荐系统, 市场分析, 协同过滤, 项目推荐。

1. 引言和相关工作

计算物品的相似性是现代推荐系统的一个关键构建模块。虽然许多推荐算法都集中在同时学习用户和物品的低维嵌入，但计算物品相似性本身就是一个目的。项目相似性被在线零售商广泛用于许多不同的推荐任务。本文处理的是通过在低维空间中嵌入物品来学习物品相似性这一被忽视的任务。

基于项目的相似性被在线零售商用于基于单一项目的推荐。例如，在Windows 10应用程序商店中，每个应用程序或游戏的详细信息页面包括一个题为 "人们也喜欢 "的其他类似应用程序的列表。如图1所示，这个列表可以扩展为与原始应用相似的项目的整页推荐列表。类似的推荐列表仅仅基于与单一项目的相似性，存在于大多数在线商店，例如亚马逊、Netflix、Google Play、iTunes商店和许多其他商店。

单一物品推荐与更 "传统 "的用户对物品推荐不同，因为它们通常是在用户对特定物品的明确兴趣和明确的用户购买意图的背景下显示的。因此，基于物品相似性的单品推荐通常比用户对物品的推荐有更高的点击率（CTR），从而负责更大的销售或收入份额。

基于物品相似性的单品推荐也被用于其他各种推荐任务。在 "糖果等级 "中，类似商品（通常价格较低）的推荐会在付款前的结账页面上提出。在 "捆绑式 "推荐中，一些商品被分组并被推荐到一起。最后，物品的相似性在网上商店中被用来进行更好的探索和发现，并改善整个用户体验。一个通过为用户定义松弛变量来隐性学习物品之间联系的用户-物品CF方法，不太可能比直接学习物品关系的方法产生更好的物品表示。

项目相似性也是基于项目的CF算法的核心，该算法旨在直接从项目-项目关系中学习表示。有几种情况需要基于项目的CF方法：在大规模的数据集中，当用户数量明显大于项目数量时，仅对项目建模的方法的计算复杂度明显低于同时对用户和项目建模的方法。例如，在线音乐服务可能有数以亿计的注册用户，但只有几万个艺术家（项目）。

在某些情况下，用户与项目的关系是不可用的。例如，今天的网上购物有很大一部分是没有明确的用户识别过程的。相反，可用的信息是每个会话。将这些会话视为 "用户 "将是非常昂贵的，而且信息量也较小。

语言任务的神经嵌入方法的最新进展极大地推进了最先进的NLP能力。这些方法试图将单词和短语映射到一个低维向量空间，以捕捉单词之间的语义关系。具体来说，Skip-gram with Negative Sampling（SGNS），也被称为word2vec，在各种NLP任务中创造了新的记录，其应用已经扩展到NLP以外的其他领域。

在本文中，我们建议将SGNS应用于基于项目的CF。受其在其他领域的巨大成功的激励，我们建议对SGNS进行少量修改可能会捕获协作过滤中不同项目之间的关系。

2.带负采样的SKIP-GRAM

带有负采样的SKIP-GRAM（SGNS）是一种由Mikolov等人介绍的神经词嵌入方法。该方法旨在找到单词表示形式，以捕获单词与句子中其周围单词之间的关系。在本节的其余部分，我们将简要介绍SGNS方法。

给定一个有限词汇表中的单词的序列，Skip-gram目标旨在最大化以下公式：

其中c是上下文窗口大小（可能取决于），而 是softmax函数：

其中和是分别对应于单词的目标表示和上下文表示的潜矢量，然后根据数据集的大小凭经验选择参数m。使用式（2）由于计算复杂性而不切实际，是词汇量W的线性函数，通常大小为。

3. ITEM2VEC –用于项目相似性的SGNS

在CF数据的上下文中，这些项目作为用户生成的集提供。请注意，有关用户和一组项目之间的关系的信息并非始终可用。例如，可能给我们一个商店收到的订单的数据集，而没有有关下订单的用户的信息。换句话说，在某些情况下，多组项目可能属于同一用户，但是未提供此信息。在第4节中，我们提供实验结果，这些结果表明我们的方法也可以处理这些情况。我们建议将SGNS应用于基于项目的CF。一旦我们意识到单词序列等同于一组项目，就可以将SGNS应用于CF数据非常简单。因此，从现在开始，我们将交替使用术语“单词”和“项目”。通过从序列移动到集合，空间/时间信息会丢失。我们选择丢弃此信息，因为在本文中，我们假设一个静态环境中，共享相同集合的项目被认为是相似的，而不管用户以什么顺序/时间生成它们。这种假设在其他情况下可能不成立，但是我们将这些情况的处理置于本文的范围之外。由于我们忽略了空间信息，因此将具有相同集合的每对项目视为一个积极的例子。这意味着根据设置的大小确定的窗口大小。具体来说，对于给定的一组项目，从等式中得出目标。 将（1）修改如下：

另一种选择是将目标保持在等式（1）中保持原样，并在运行时随机播放每组项目。在我们的实验中，我们观察到两个选项的执行效果相同。 其余过程与第2节中描述的方法相同。我们将描述的方法命名为item2vec。 在这项工作中，我们使用作为第i个项目的最终表示形式，并且通过余弦相似度来计算一对项目之间的相似度。 其他选项是使用，添加成分或串联的。 请注意，最后两个选项有时会产生更好的表示效果

附录C