硕士学位论文

（学术学位）

Spark 负载均衡及随机森林算法优化研究

学位申请人：张占峰

指 导 教 师：王素贞　教授

学 科 专 业：计算机应用技术

学 位 类 别：工学硕士

培 养 学 院：信息技术学院

答 辩 日 期：二〇二〇年五月

中图分类号：TP39　　密　　级：公开

UDC：004　　学校代码：11832

Spark 负载均衡及随机森林算法优化研究

The Optimization research of Spark Load Balancing and Random Forest Algorithm

学位申请人：张占峰

指 导 教 师：王素贞　教授

学 科 专 业：计算机应用技术

学 位 类 别：工学硕士

培 养 学 院：信息技术学院

答 辩 日 期：二〇二〇年五月

学位论文原创性声明

本人所提交的学位论文《Spark 负载均衡及随机森林算法优化研究》，是在导师的指导下，独立进行研究工作所取得的原创性成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中标明。

本声明的法律后果由本人承担。

论文作者（签名）：张占峰　　指导教师确认（签名）：王景亚

2020 年 5 月 21 日　　2020 年 5 月 21 日

学位论文版权使用授权书

本学位论文作者完全了解河北经贸大学有权保留并向国家有关部门或机构送交学位论文的复印件和磁盘，允许论文被查阅和借阅。本人授权河北经贸大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其它复制手段保存、汇编学位论文。

（保密的学位论文在_____年解密后适用本授权书）

论文作者（签名）：张占峰　　指导教师（签名）：王景亚

2020 年 5 月 21 日　　2020 年 5 月 21 日

摘　要

随着信息技术的快速普及，各行各业产生并积累了海量的数据，因此如何高效地处理海量数据，从中挖掘出有价值的信息成为急需解决的重要问题。近年来，从平台方面看，Spark 作为一种基于内存计算的高效的大数据处理平台，能够较好地支撑解决大数据挖掘分析处理的一系列问题，成为了学界和产业界的研究热点；从算法方面看，基于 Spark 平台的数据挖掘算法优化也是一个研究热点，随机森林算法是数据分类方法中的典型算法，因其较好的分类性能被广泛的应用，因此研究基于 Spark 的随机森林算法具有理论意义和实用价值。本文对于 Spark 平台及基于 Spark 平台的随机森林分类算法进行了相关研究，主要包括以下两个方面的内容：

（1）Spark 负载均衡优化研究

Spark 是一种基于内存计算的高效大数据处理平台，集群的负载均衡情况对于集群的运算效率具有重要影响。但其默认的任务调度策略在 Spark 集群下未考虑到节点的可用资源及节点当前负载的具体情况，因此在进行任务调度时可能会导致各个节点负载不均衡，进而影响集群的任务处理效率。针对于 Spark 的负载不均衡问题，本文提出一种基于 Spark 集群的自适应任务调度策略用于实现 Spark 集群的负载均衡优化。该策略根据节点的计算资源及负载的实际情况，使用蚁群模拟退火融合算法的启发式算法，对 Spark 集群的任务调度策略进行优化，实现任务的合理分配，以达到负载均衡优化的目的，从而提升集群的任务处理效率，并通过实验验证了本文所研究的对于 Spark 集群负载均衡优化的有效性。

（2）基于 Spark 的随机森林算法优化研究

在进行数据分析时，数据中往往包含一些冗余特征，随机森林算法在处理数据时，采用随机选择特征的方式形成特征子空间，而该方式在生成特征子空间时无法区分这些冗余特征，故而会影响随机森林算法的分类准确率。针对此问题，本文基于 Spark 平台对随机森林算法进行了优化。优化后的随机森林算法通过计算特征重要性进行强弱相关特征区分，随后采用分层抽取特征的方式形成特征子空间，以此提高随机森林算法整体的分类准确率。随后本文在 Spark 平台对优化后的随机森林算法进行了并行化并对改进后的算法分类准确率进行了验证。最后将优化后的随机森林算法应用于信用评估数据集，

并通过结果验证了改进后的随机森林算法能够有效提升信用评估的准确率。

关键词：Spark；负载均衡；启发式算法；随机森林算法；特征子空间

Abstract

With the rapid popularization of information technology, a large amount of data has been generated and accumulated in various industries. Therefore, how to efficiently process the large amount of data and extract valuable information from it has become an important problem to be solved urgently. In recent years, from the perspective of platform, Spark, as an efficient big data processing platform based on in-memory computing, can better support and solve a series of problems of big data mining analysis and processing, has become a research hotspot in academia and industry. On the other hand, from the perspective of algorithm, the optimization of data mining algorithm based on Spark platform is also a research hotspot. Random forest algorithm is a typical algorithm in data classification methods, which is widely used because its better classification performance, so research on random forest algorithm based on Spark has theoretical significance and practical value. In this paper, the Spark platform and the random forest classification algorithm based on Spark platform are researched, mainly including the following two aspects:

(1) The optimization research of Spark load balancing:

Spark is an efficient big data processing platform based on in-memory computing, the load balancing of cluster has an important influence on the computing efficiency of cluster. However, the default task scheduling policy in Spark cluster does not consider the available resources of the nodes and the specific situation of the current load of the nodes, so it may lead to unbalanced load of each node in the process of task scheduling, thus affecting the task processing efficiency of the cluster. Aiming at the unbalanced load problem of Spark, this paper proposes an adaptive task scheduling policy based on Spark cluster to realize the load balancing optimization of Spark cluster. The strategy according to the computing resources of nodes and the actual situation of the load, uses the heuristic algorithm of ant colony simulated annealing fusion algorithm to optimize the task scheduling strategy of Spark clusters. And achieve the goal of load balancing and optimization by reasonably distribute the tasks, so as to improve the task of cluster processing efficiency. Finally, the effectiveness of the Spark

cluster load balancing optimization studied in this paper is verified by experiments.

(2) The optimization research of random forest algorithm based on Spark:

When doing data analysis, data often contain some redundant features. And random forest algorithm adopts the method of random selection of features to form feature subspace when it is used to do the data mining, which cannot distinguish these redundant features when generating feature subspace, thus affecting the classification accuracy of random forest algorithm. Aiming to solve this problem, this paper optimizes the random forest algorithm based on Spark platform. The optimized random forest algorithm differentiates the strong and weak correlation features by calculating the importance of the features, and then forms the feature subspace by means of hierarchical feature extraction, so as to improve the overall classification accuracy of the random forest algorithm. Subsequently, this paper parallelizes the optimized random forest algorithm on Spark platform and verifies the classification accuracy of the improved algorithm. Finally, the optimized random forest algorithm is applied to the credit evaluation data set, and the results verify that the improved random forest algorithm can effectively improve the accuracy of credit evaluation.

Keywords: Spark; Load balancing; Heuristic algorithm; Random forest algorithm; Feature subspace

目 录

1 绪论

1.1 研究背景

当今社会，信息技术的发展带来了互联网在各行各业的迅速普及，同时促进了国家信息网络，个人商业网络的急剧扩张，随之而来的是每天产生的 TB 甚至 PB 级别的海量数据。“大数据”就是伴随着信息数据爆炸式增长和网络计算技术迅速发展而兴起的一个新型概念。由于大数据作为可以被二次甚至多次开发的新的“资源”已经渗透进了医疗、教育、商业等多个行业，因此带来了中国大数据产业的飞速发展。《2019 中国大数据产业发展白皮书》显示，2018 年，中国的大数据产业规模已经超过 4000 亿元，达到 4384.5 亿元之巨，预计 2021 年，这一数字将超过 8000 亿元[1]，中国正在飞速进入“大数据时代”。

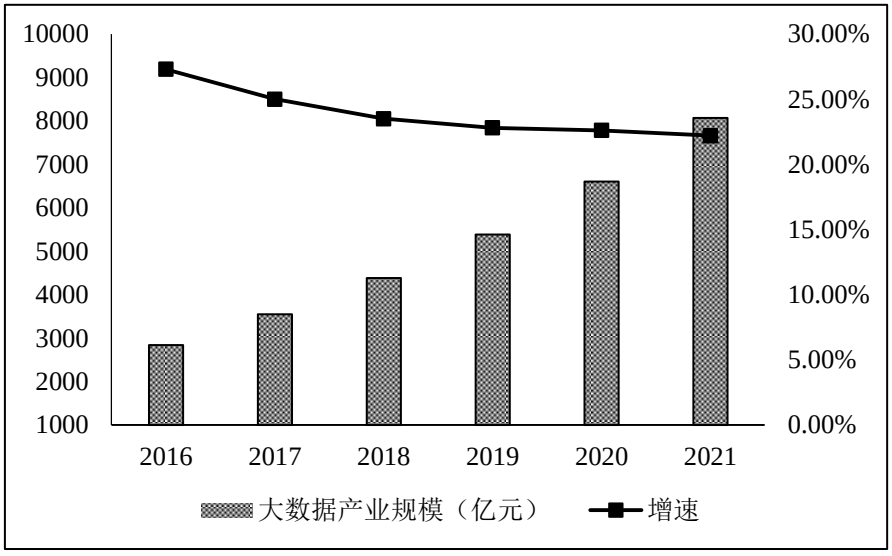

图 1-1 中国大数据产业规模

大数据时代的到来推动了云计算和大数据技术的快速发展。HDFS、MapReduce、HBase 等的逐渐诞生，标志着基于 Hadoop 的生态圈逐步形成。但是随着数据量的快速增长以及用户需求的不断增加，使得人们对于数据处理效率的要求也越来越高，在此背景下，基于内存计算的 Spark 计算框架应运而生，并迅速成为人们关注的热点[2]。作为由美国加州大学伯克利分校 AMP Lab 开发的一款分布式计算框架，Spark 的计算思想与 Hadoop 十分相似，但是不同于 Hadoop 计算框架，Spark 使用内存处理中间计算结果，

这种方式使得 Spark 更加适合于迭代计算和数据的实时处理[3]。

Spark 高效的计算性能使其得到了广泛的应用[4]：淘宝技术团队利用 Spark 的计算性能的优势取代了部分 MapReduce 并已经应用于淘宝的推荐系统当中；美团技术团队则借助 Spark 快速迭代的优势将其应用于美团的交互式用户行为分析系统；360 技术团队使用 Spark 来进行数据的实时分析处理并应用于实时广告重定向当中；腾讯技术团队则通过 Spark 对于数据进行实时采集、实时分析，进而实现了精准推荐。

1.2 研究目的及意义

Spark 具有强大的数据处理能力及较高的可用性，但是在进行实际数据分析时，集群的配置参数、负载均衡程度，合理的任务调度方式等都是会对计算效率产生影响的至关重要的因素。另一方面，在使用 Spark 平台进行数据处理时，良好的数据挖掘算法对于提升处理效果同样有着重要的意义。随机森林算法由于其在面对噪声和异常值数据集时较好的表现，且能够一定程度上克服过拟合的问题[5]，因此在数据挖掘领域得到了广泛的应用。鉴于此情况，本文对 Spark 平台负载均衡及基于 Spark 平台的随机森林算法进行了相关研究。

当 Spark 出现集群负载不均衡情况时，会导致资源的闲置浪费，影响计算效率，因此集群的负载均衡是 Spark 处理计算任务时的重要目标之一。平台的任务调度策略对于集群负载是否均衡具有重要影响。在使用 Spark 平台进行数据处理时，随着时间的推移，会导致 Spark 集群各节点的当前可用资源不均等，默认的 Spark 任务调度策略没有考虑到节点的负载及当前可用资源的具体情况，因此设计一种合适的任务调度策略对 Spark 集群的负载均衡优化以及提升集群性能十分重要。

随机森林算法具有对数据集的适应能力强、能够有效避免过拟合等优点，因此在很多领域有着广泛的应用，包括信用评估[6]-[8]、图像分类[9]-[10]、文本分类[11]等。但是随机森林算法在处理数据时，由于数据的许多特征与类别的相关性较弱，因此随机森林算法简单抽样选择特征生成特征子空间的方式会影响随机森林算法的分类准确率，故在进行特征子空间生成时对特征的选择方式进行优化对于提升随机森林算法的分类能力具有重要的意义。

1.3 国内外研究现状

1.3.1 负载均衡研究现状

并行计算的思想已经广泛应用于大数据的分析处理，而并行计算集群的负载均衡程度是影响集群整体性能的关键因素之一。Spark 作为当今流行的基于内存的并行计算框架，如何能够较好地保证集群的负载均衡进而有效提升集群的数据处理效率已经成为了人们关注的问题，因此已经有国内外学者对于 Spark 的集群负载均衡进行了相关研究。目前对于 Spark 负载均衡的研究主要有以下两个方向：

（1）基于数据均衡方向的研究：

Tang 等[12]对于 Spark 的中间数据负载均衡执行策略进行了研究，提出了对于中间数据进行分割和组合的 SCID 算法，通过对<key,value>进行合理分割组合来实现数据负载均衡，进而实现 Spark 的负载均衡。李巧巧[13]针对 Spark 的任务中间结果造成的数据不均衡情况，重新定义了任务细粒度，并结合 BP 神经网络，对任务进行动态预测，对数据倾斜的任务进行动态切割以使 Spark 各阶段节点负载均衡。黄超杰[14]提出了一种数据均衡分配算法 MRFair，该算法针对 Spark 平台中的部分任务计算数据量过大的问题，通过预估剩余任务的所需要的计算时间，将所需时间最长的未处理的数据进行重新分配，实现数据均衡。Liu 等[15]则基于系统抽样的样本进行中间数据的特征预测，用于进行数据的动态分配，达到平衡 Reducer 工作量的目的，实现数据均衡。

（2）基于任务调度策略方向的研究：

Aljawarneh 等[16]在 Spark 中引入了一套自适应负载平衡的框架，根据负载的具体情况，以自适应的方式调整框架执行，提高了集群的整体性能。杨志伟[17]则针对异构环境下的 Spark 负载均衡问题，提出了一种自适应任务调度策略，该策略通过获取节点的当前可用资源来给节点赋予权重，随后根据权重高低选择任务执行节点，从而有效提高了 Spark 集群的负载均衡程度，但是该算法没有考虑到不同任务的具体情况。Liang 等[18]则提出了一种基于节点负载的动态任务调度算法，该算法通过节点当前的负载信息给动态任务分配提供依据，达到将任务分配给空闲节点，降低单个节点负载过重的目的，但是该算法没有考虑到任务不同阶段所需资源不同的问题。Xu 等[19]则提出了一种基于启发式方法的大数据平台异构感知任务调度系统 RUPAM。该系统在考虑底层节点资源的

异构性以及任务不同阶段对于资源需求的异构性的情况下，通过获取任务特征及计算节点实际情况来进行任务执行节点分配，提高了集群性能。Du 等[20]则提出了一种基于任务完成时间感知的任务调度方法，该方法通过深度学习进行任务完成时间预测，并在此基础上基于每个任务的数据大小，数据节点的异构性，每个任务的计算复杂性以及网络传输情况，进行任务合理分配，以此来平衡负载，实现计算资源的高效利用，提升计算效率。

除了以上两个方面的研究以外，由于 Spark 使用了与 Hadoop 相似的计算思想，因此针对于 Hadoop 的负载均衡研究同样具有一定参考价值。目前针对于 Hadoop 的负载均衡已经有了许多积累。在数据的均衡放置方面，Fan 等人[21]提出了一种基于磁盘利用率和服务阻塞率模型的 HDFS 自适应反馈负载均衡算法，优化了数据均衡。刘党朋[22]则提出了一种节点负载评估模型并在此基础上提出了改进的数据副本放置策略，提高了数据均衡程度。在任务动态调度方面，刘盼红[23]针对于异构环境下集群负载不均衡的问题，提出了基于负载均衡度量函数的粒子群调度算法，对任务进行动态调度。冯亮亮[24]针对 MapReduce 在任务调度过程中的负载不均衡问题，提出了基于任务特点及节点性能的动态调度算法，提升了集群负载均衡程度。姜淼[25]则针对 Hadoop 中的调度算法无法根据集群的具体状态进行调整的问题，根据作业的资源需求和集群的资源配置情况结合分类器进行任务调度，提高了集群性能；于磊春[26]则结合节点负载信息，通过动态化轻量级的划分策略，来实现 Reduce 端的动态化负载均衡。以上研究均在一定程度上对于 Hadoop 集群在执行任务时出现的负载不均衡问题进行了优化，可以给 Spark 的相关研究提供有效的参考。

1.3.2 随机森林算法及基于 Spark 的随机森林算法研究现状

关于随机森林算法的研究主要有以下方面：

李贞贵[27]对于分类算法的分类间隔与泛化能力进行了研究，提出了一种基于分类间隔加权的随机森林修剪算法，在提升算法的分类效果的同时，还能减少森林的规模，降低了算法的复杂度，但是并未考虑到冗余特征对于特征子空间生成时的影响。Robnik 等[28]则针对随机森林算法在投票时的不足，提出了间隔的加权投票方法，提升了算法的整体分类效果。Amaratunga 等[29]则针对特征选择中的不足进行了研究，在特征选择时

采用了加权随机抽样来过滤掉具有很少信息的特征，提高了特征子空间的有效性。马春来等[30]则针对随机森林算法在处理高维且包含冗余特征时的不足，对特征值赋予权重并以此构造特征子空间，从而提升算法的分类效果。Joaqu ń 等[31]则在传统随机森林算法的基础上提出了 RCRF 算法，该算法改进了传统随机森林的拆分标准，增强了不同决策树之间的差异性，同时对于噪声数据具有更强的鲁棒性。

基于 Spark 的随机森林算法方面的有关研究主要有以下方面：

王雪[32]针对随机森林算法在面对高维特征数据时的问题，提出基于相似性的特征选择方法来进行优化。该方法通过计算特征之间的相似性进行特征选择，去除冗余特征，生成分类效果更好地特征子空间，提升了算法的分类效果。罗元帅[33]则基于差别矩阵进行特征选择来生成特征子空间，并使用 F1 值进行决策树投票重要度加权，优化了算法的分类效果。王日升[34]基于 Spark 提出了一种基于分类精度和相似度的改进的随机森林算法，该算法首先对随机森林模型中的决策树进行评判，得到分类效果较好地决策树，随后对得到的决策树集合进行相似度计算，将相似度较好的决策树进行聚类并选出该聚类中分类效果最好的决策树作为代表用于组建最终的随机森林模型，使得随机森林算法的分类准确率得到了提升。Xu[35]则基于 Spark 提出了一种基于 Fayyad 边界点原理的改进的随机森林算法（FRF），以解决经典随机森林算法在连续属性离散化过程中的缺点，提升了算法性能。Man[36]则基于 Spark 对随机森林算法的节点数据分割方式进行了相关研究，通过在随机森林算法中的 ID3 和 CART 中重新组合属性拆分模式，获取新的拆分规则，然后通过自适应参数选择，获得了最优的属性选择，以此来实现分类效果的提升。

1.4 本文结构安排

本文共分为五个章节，具体结构安排如下：

第一章，绪论。本章首先介绍了论文的研究背景，随后对于论文的研究目的和意义进行了阐述，最后对于 Spark 负载均衡及随机森林算法的国内外学者的研究现状进行了综述分析。

第二章，相关技术分析。本章对于论文中涉及到的相关技术理论进行了分析，首先对 Spark 生态系统进行了介绍；随后对 Spark 的相关核心技术进行了简要说明，包括 Spark 运行架构、弹性分布式数据集及 Spark 的作业调度机制等。随后对于启发式算法进行了

介绍，最后对随机森林算法及特征选择算法进行了相关说明。

第三章，Spark 负载均衡优化研究。本章首先对 Spark 中的负载均衡问题进行了分析，进而提出了改进策略，随后对于用于优化集群负载均衡策略的基于蚁群模拟退火融合算法的任务调度算法进行了说明，最后进行了实验验证并对实验结果进行了分析比较。

第四章，基于 Spark 的随机森林算法优化研究。本章首先对于随机森林算法的问题进行了分析，并提出了优化方法。然后对优化方法中的基于特征重要性的特征子空间生成策略进行了详细阐述，随后对于优化后的随机森林算法的并行化设计进行了说明，最后通过实验验证了算法改进的有效性。

第五章，总结与展望。本章节总结了本文的相关工作，对本文工作中的不足之处进行了说明，并对未来的研究工作进行了展望。

2 相关技术分析

2.1 Spark 生态系统

以 Spark Core 为核心的 Spark 平台生态系统十分庞大，不仅支持包括 Standalone、YARN、Mesos 在内的多种部署模式，而且能够从包括 HDFS、HBase、Hive 等在内的多种数据源读取数据。而且 Spark 的设计涵盖了多个处理组件以方便用户进行相关的数据处理。下面本节将对 Spark 生态系统进行介绍。

Spark 生态系统如图 2-1 所示。

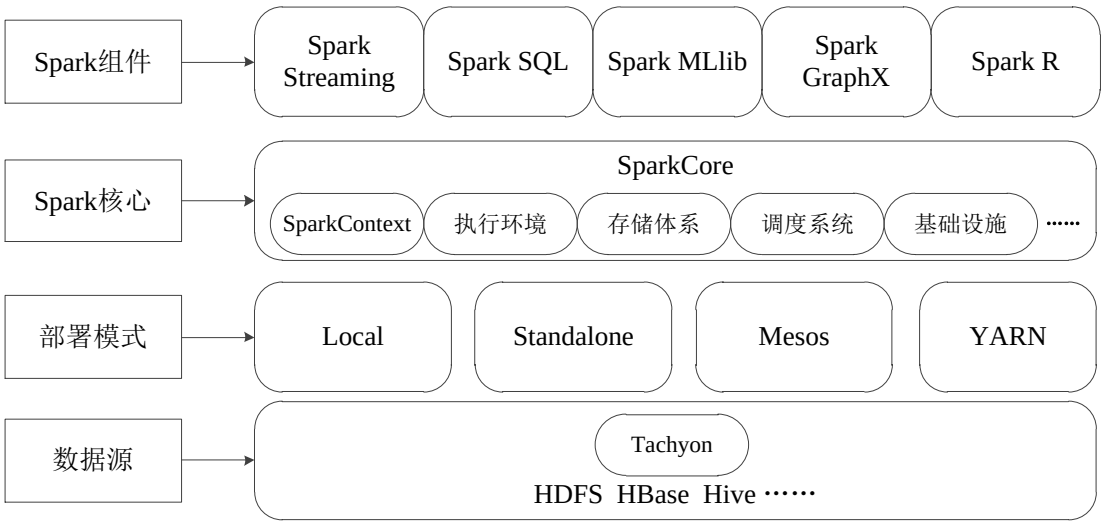

图 2-1 Spark 生态系统

2.1.1 Spark 组件及核心

Spark 组件中主要包括 Spark Streaming、Spark SQL、Spark MLlib、Spark GraphX 以及 Spark R 等。

1. Spark Streaming:

Spark Streaming 是 Spark 中的核心 API 的扩展，支持可伸缩、高吞吐量的实时数据流处理，并且具有容错机制[37]。在 Spark Streaming 中，数据可以从许多来源获取，如 Kafka、Flume、Kinesis 或 TCP sockets 等。在此基础上，Spark Streaming 支持用户使用复杂的算法处理数据，并且支持许多高级函数，如 map、reduce、join 等。经过 Spark

Streaming 处理后的数据可以推送到文件系统、数据库等多种数据存储系统当中。此外，还可以将 Spark 的机器学习和图形处理算法应用于 Spark Streaming。

2. Spark SQL

Spark SQL 是 Spark 中一个用于结构化数据处理的模块[38]。Spark SQL 提供的接口为 Spark 提供了包括有关数据结构在内的诸多信息，而 Spark SQL 则使用这些信息来执行相关的优化。

用户可以使用包括 SQL 和 Dataset API 在内的多种方法与 Spark SQL 进行交互。在进行计算时，Spark SQL 并不依赖于使用哪种 API 或语言来进行计算，因此用户可以很容易地在不同的 API 之间进行切换。

Spark SQL 可以从包括本地文件系统，Hive，HBase 等在内的多种数据源读取文件。并且 Spark SQL 支持通过 DataFrame 接口对各种数据源进行操作。

3. Spark MLlib：

Spark MLlib 是 Spark 的机器学习算法库。它的主要目的是将传统的机器学习算法在 Spark 中进行并行化，使得用户使用更为方便。同时还提供了许多工具包，主要有以下几种：

（1）机器学习算法：在该工具包中，Spark MLlib 提供了包括分类，聚类，回归，协同过滤在内的许多机器学习算法供用户使用。

（2）特征工程：提供了特征选择、提取、降维及特征变换在内的多种算法。比如：卡方检验，主成分分析（PCA）等[39]。

（3）持久化：用于保存和加载相关的机器学习模型及算法等。

（4）数据工具：如线性代数，统计模型，数据处理等工具。

（5）Pipeline：该部分提供了用于构建，评估和调整机器学习管道的工具。

Spark ML 也是 Spark 的一个机器学习算法库[40]。但是与 Spark MLlib 基于 RDD 的数据格式不同，Spark ML 使用的是 DataFrame 数据格式。此外，在 Spark MLlib 中含有的任何库和函数在 Spark ML 中都可以找到。

4. Spark GraphX:

GraphX 是 Spark 上用于进行图形计算的组件，是对 Spark RDD 的一种通过引入图形的抽象扩展[41]。如同 Spark Streaming，Spark SQL 一样，它也继承了 RDD API。它提供了包括 subgraph，joinVertices 和 aggregateMessagesTable 在内的多种运算符及相应操

作算子，提供了 PageRank、TriangleCount、最短路径等常见图算法模型。Spark GraphX 可以应用在很多领域[42]，如个性化推荐、欺诈监测、最短路径、社区检测、主题建模等。

5. Spark R

Spark R 是 R 软件包，它提供了方便的前端用于在 Spark 上应用。Spark R 提供了分布式数据帧实现，支持在大型数据集上进行诸如选择，过滤，聚合等操作。此外，Spark R 还支持使用 Spark MLlib 进行分布式机器学习。

6. Spark 核心

Spark 核心（Spark Core）是 Spark 的底层通用执行引擎，为其他所有功能组件服务，在其内部定义了包括 Spark Context 等 Spark 的基本功能，同时定义了包括任务调度机制在内的许多框架负责保证 Spark 基本功能的运行。

2.1.2 部署模式

Spark 支持多种部署模式，其中主要有以下几种：

Local（本地模式）：常用于本地开发测试，包括 local 单线程和 local-cluster 多线程；其中 local 单线程不需要启动 Spark 的相关 Master、Worker 守护进程；local-cluster 多线程则是伪分布式集群模式，在该模式下，会在单机启动多个进程来模拟集群下的分布式场景。

Standalone 模式：该模式使用 Spark 自身的的资源管理系统，是可以单独部署到集群中的独立模式。在该模式下集群在执行应用程序时必须先启动 Master 与 Worker 两者的守护进程。而且该模式下 Master 和 Worker 均可以有多个，但是 Master 只能有一个处于激活状态。此外，在该模式下可以使用 Zookeeper 解决 Master 的单点故障问题。本文即是在此模式下进行了相关研究。

Spark on Mesos 模式：Spark 在该模式下运行相比于在 Yarn 模式下运行要更加灵活。在该模式下运行时，资源可以在 Spark 以及其他框架下动态划分，而且可以在多个 Spark 实例同时部署的情况下调整各个实例之间的资源分配。目前该模式下的调度模式为类似云计算的按需分配计算资源的细粒度模式。但是在细粒度模式下会导致作业的执行时间被延长。

Spark on YARN 模式：在该模式下，集群的资源和任务管理是通过 Yarn 实现的。目

前在该模式下只支持静态分配资源，即在进行计算之前提前申请计算资源，而在运行时不再进行资源的动态分配，因此 Spark on YARN 模式下在进行计算时很容易导致计算资源被浪费。

2.2 Spark 相关技术分析

2.2.1 Spark 运行架构

Spark 在运行时采用的是 Master/Worker 主从架构，并且包含了包括 Driver、ClusterManager 在内的许多角色，如图 2-2 所示，而这些不同角色在执行计算任务时有各自不同的职责安排。

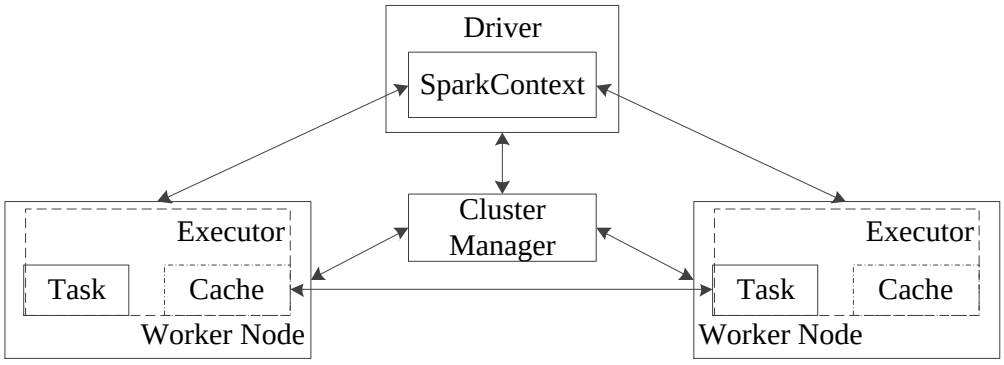

图 2-2 Spark 整体架构

Driver 是 Spark 应用程序的入口，负责创建 SparkContext，将用户编写的程序转化为可在集群上进行实际运算的 Spark 可执行程序。ClusterManager 则负责在程序执行过程中获取计算资源并将资源分配给相应的应用程序。Worker Node 则是 Spark 集群中的任务执行节点，在 Standalone 调度模式下时是指通过 slave 文件配置的 Worker 节点，在 Yarn 模式下则是 NodeManager 节点。Executor 为负责执行任务的分布式代理，是工作节点上的应用程序启动的进程，该进程运行任务并将数据存储在内存或磁盘中[43]。

2.2.2 弹性分布式数据集（RDD）

RDD 是 Spark 运行和计算的核心，是可并行操作的具有容错性的元素集合。整个 Spark 都围绕 RDD 进行展开。在 Spark 中 RDD 有两种创建方式，第一种是计算程序中

产生的数据集合，该方法是在已有的数据集合上调用 SparkContext 的 parallelize 方法创建的；复制这些创建的数据集合即可以形成能够并行操作的分布式数据集。第二种则是从外部存储系统中获取的数据集合，外部存储系统包括本地文件系统，HDFS，Cassandra，HBase，Amazon S3 等。

在 RDD 中支持两种类型的操作：Transformation 与 Action，其中 Transformation 是根据现有操作创建新的数据集，而 Action 则是在数据集上进行计算后将其返回给可执行程序。在 Spark 中，所有的 Transformation 操作都是惰性操作，不会立即计算出结果，反而只会对应用于某些数据集的转换进行记忆，只有当 Action 操作要求将结果返回给可执行程序时才进行转换。这种设计大大提升了 Spark 的运行效率。

RDD 之间通过血缘产生“宽依赖”与“窄依赖”两种依赖关系，如图 2-3 所示。其中宽依赖指父 RDD 的每个分区可以被多个子 RDD 的分区所依赖，窄依赖则是父 RDD 的每个分区最多被其子 RDD 的一个分区所依赖。

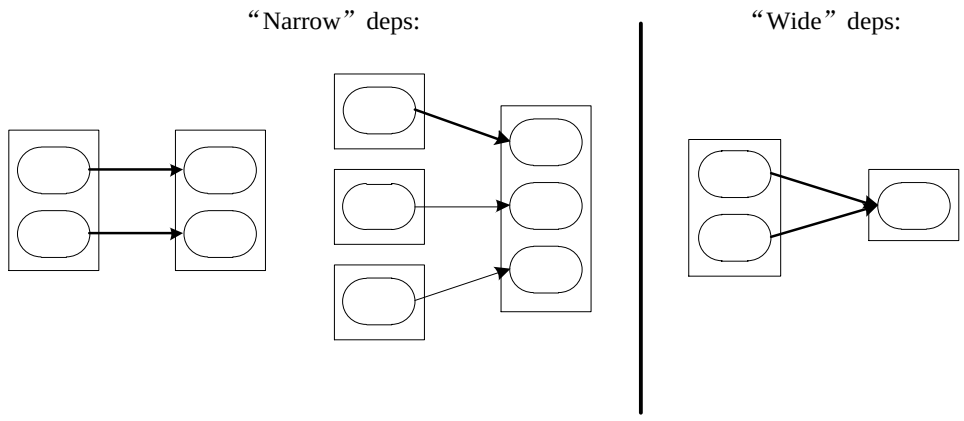

图 2-3 RDD 的宽依赖与窄依赖

默认情况下，每次对 RDD 执行操作时，都可能会重新计算每个转换后的 RDD。但是，用户可以将 RDD 保留在内存中，以便下次查询时可以更快地进行访问。此外还支持将 RDD 持久存储在磁盘上，或在多个节点之间复制。

在 Spark 中，最重要的功能之一是跨操作将 RDD 持久化（或缓存）在内存中。当需要保留 RDD 时，每个节点都会将其计算的所有分区存储在内存中，并在该数据集或其衍生数据集上的其他操作中重用它们。这样可以提升效率。缓存是用于迭代算法和快速交互使用的关键工具。用户可以使用 persist()或 cache()方法将 RDD 标记为要保留，

当第一次在操作中对其进行计算时，它将被保存在节点上的内存中。Spark 的缓存具有容错机制，如果 RDD 的任何分区丢失，它将使用最初创建它的 Transformation 操作自动重新计算。

2.2.3 Spark 作业调度机制

默认情况下，Spark 的作业调度程序以 FIFO 方式运行作业。每个作业都分为不同阶段，第一个作业在所有可用资源上都具有优先级，而其各个阶段都有要启动的任务，随后第二个作业具有优先级，依此类推。队列不需要使用整个集群，以后的作业可以立即开始运行，但是如果队列开头的作业很大，则以后的作业可能会大大延迟。

而在 Spark 中的具体任务分配是通过有向无环图实现的。在 Spark 中，各个 RDD 之间存在着依赖关系，这些依赖关系形成了有向无环图 DAG，DAGScheduler 对这些根据依赖关系形成的 DAG 进行 Stage 划分，并且根据 Stage 来生成 Taskset，并将其提交给 TaskScheduler。在 Spark 中，具体的任务调度由 TaskScheduler 负责，任务的计算则在 Worker 节点中的 Executor 中进行。

在该流程中，DAGScheduler 主要工作如下：

（1）DAGScheduler 对 DAG 有向无环图进行 Stage 划分。

（2）记录哪个 RDD 或者 Stage 输出被物化，方便之后的计算。

（3）重新提交 Shuffle 输出丢失的 Stage（Stage 内部计算出错）给 TaskScheduler。

（4）将 Taskset 传给底层调度器。

TaskScheduler 主要工作如下：

（1）为每一个 Taskset 构建一个 TaskSetManager 实例，用于管理这个 Taskset 的生命周期。

（2）提交 Taskset（一组 task）到集群运行并监控其状态。

（3）推测执行，碰到计算缓慢任务需要放到别的节点上重试。

（4）重新提交 Shuffle 输出丢失的 Stage 给 DAGScheduler。

2.3 启发式算法

启发式算法是用于解决在商业，工程，工业和许多其他领域中出现的复杂优化问题

的近似搜索方法的新发展，是一种确定优化问题的最优或者近似最优解的过程。

自上世纪七十年代以来，研究人员已经开发了许多启发式算法并将其应用于不同的优化问题，通常可分为以下几类：

1.进化算法：这些算法模拟生物进化的概念，如遗传算法（GA），差分进化算法（DE），进化策略（ES），进化规划（EP）以及生物地理学优化算法（BBO）等。

2.群智能算法：这些算法模仿去中心化与自组织系统的过程，通常是对生物界中种群行为的一种模仿，例如蚁群算法（ACO），粒子群优化算法（PSO），人工蜂群（ABC）算法，布谷鸟搜索（CS）算法，萤火虫算法（FA），蝙蝠算法（BA），海豚回声算法（DE）以及人工鱼群算法（AFSA）等。

3.物理算法：这些算法的思想通常来自物理学现象或者物理学定律，模拟退火算法（SA）是该类启发式算法的典型示例。

4.人工神经网络：神经网络算法作为一种模仿人类大脑神经进行分析处理过程的算法，通过不同神经元之间的竞争与合作，来实现选择和变异的过程。人工神经网络算法应用广泛，且兼容性较好，可以与其他启发式算法结合使用。

启发式算法，各有优劣，多数都需要面对容易陷入局部最优的问题，而且不同启发式算法的参数设置情况各不相同，因此将不同启发式算法相结合，对单个启发式算法的不足进行改善，是提升启发式算法效果的良好途径。本文使用启发式算法进行负载均衡的相关研究。

2.4 随机森林算法分析

2.4.1 决策树

随机森林算法是一种以决策树为基分类器的集成算法，在随机森林算法中，随机森林由多棵决策树组成。决策树作为一种分类器，类似于倒立的树状结构，从根节点开始，根据分裂规则进行节点分裂，直至决策树生成完毕。在决策树中，叶节点由数据样本根据某一特征的分裂产生，代表了一个类标号，而从根节点到叶节点的路径则表示某种决策过程[44]。

构建决策树主要由节点分裂、生成决策树及剪枝三个步骤组成。

（1）节点分裂

用于进行节点分裂的特征主要有以下三种选择方式：

① 信息增益

将数据集根据某特征划分前后的熵差定义为信息增益。当信息熵差较大时，则说明数据集的不确定性较强，因此，可以利用划分前后数据集信息熵的差值来判断当前划分的特征对于数据集划分的效果。信息增益的方法即是通过计算数据集中所有特征的信息增益，选择信息增益最大的特征对数据集进行分割。

② 信息增益比

信息增益比即是在信息增益的基础上增加了一个与数据集特征数量有关的惩罚参数。数据集中的特征数量越大，惩罚参数越小，反之，当数据集中特征数量较小时，惩罚参数则会变大。

③ 基尼系数

基尼系数为数据集中任意数据样本被错误分类的概率。基尼系数越小，数据集中样本被错误分类的概率越低，则数据集的纯度越高，反之，基尼系数越高，则说明数据集的纯度越低，样本被错误分类的概率越高。

（2）决策树生成

由于节点分裂方式的不同，因此产生了不同的决策树生成方式，其中 ID3 算法是使用信息增益选择特征进行节点分裂来生成决策树，C4.5 算法则是使用信息增益比选择特征进行节点分裂来生成决策树，而 CART 算法则是使用基尼系数选择特征进行节点分裂来生成决策树。

（3）决策树剪枝

决策树剪枝是为了防止出现过拟合现象而进行的决策树简化，通过删除部分分支，来简化决策树模型，达到防止出现过拟合现象的目的。

2.4.2 随机森林

决策树具有模型简单易于实现等特点，但是在进行分类时，易出现过拟合以及无法保证取得全局最优等缺陷，因此基于决策树算法的特点，随机森林算法应运而生。如图 2-4 所示，随机森林算法的核心思想是将多棵由样本子集训练产生的决策树进行组合以

提升算法的分类准确性，避免过拟合等现象[45]。

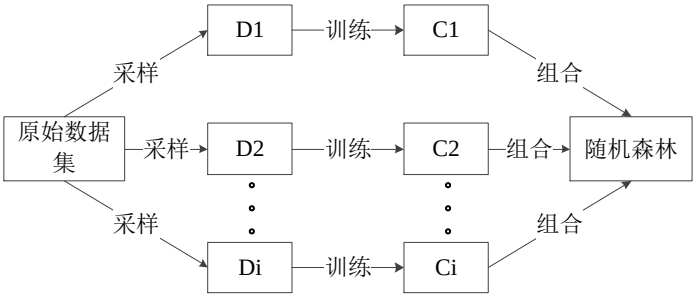

图 2-4 随机森林算法核心思想

在生成随机森林时，主要分为两个步骤，第一步为生成多棵相互独立的决策树，第二步为决策树对于分类结果进行投票并产生最终结果，具体步骤如下：

Step 1：对训练数据样本集进行有放回抽样，获取 N 个样本子集。

Step 2：获取样本子集后，进行特征子集的选择。

Step 3：在数据样本子集与特征子集的基础上训练单棵决策树。

Step 4：将训练完毕的决策树进行组合，以投票方式对数据进行分类。

算法流程如图 2-5 所示：

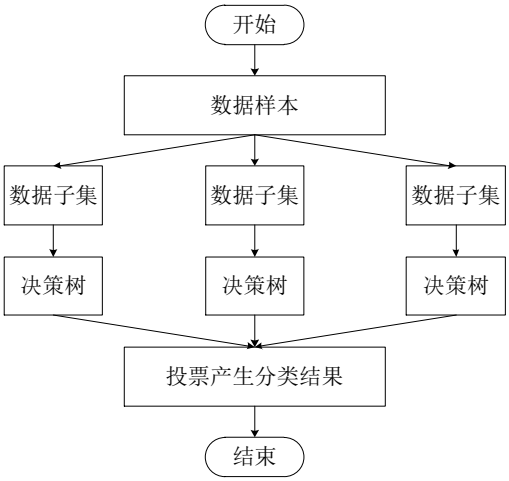

图 2-5 随机森林算法流程

随机森林算法主要解决了单棵决策树在进行分类时易出现的过拟合现象以及局部最优问题。然而，随机森林算法并非完美。它有以下缺点：

第一，包含冗余特征的高维数据会影响随机森林算法的分类准确率。

第二，无法区分单棵决策树的分类准确性，因此随机森林模型的分类准确率一定程度上会受到分类精度弱的决策树影响。

2.5 特征选择算法

特征选择又称为属性选择，是指从全部数据集中选择一个子集，使得模型效果最优。通过特征选择，可以去除数据中的冗余或者不相关的数据。例如，在一个动物分类项目中，研究人员可以使用特征选择来只选择一个或多个调查物种的特定分类信息，并消除项目中不重要的其他数据。除此之外，特征选择还能帮助研究人员解决数据中的“维数爆炸”的问题。

一个特征选择过程主要包含以下几个步骤：

Step 1.搜索特征子集；

Step 2.根据评价函数停止特征子集的搜索，产生特征子集

Step 3.通过实验数据集验证产生的特征子集的有效性。

在特征子集的搜索过程中，常用的搜索方法主要有完全搜索、启发式搜索和随机搜索三大类，其中完全搜索主要包括广度优先搜索、分支限界搜索、定向搜索以及最优优先搜索四种方法。启发式搜索则主要包括序列前向选择、序列后向选择、双向搜索、LRS搜索等多种方式。常见的随机搜索方法则主要有随机产生序列选择算法及遗传算法等。

而常见的特征选择算法可以分为 Filter 与 Wrapper 两类，Filter 特征选择算法主要有 Relief 算法、CFS 算法等，Wrapper 特征选择算法主要有 LVW 算法等。Wrapper 特征选择算法的评价函数主要是分类器精度，而常用的 Filter 特征选择算法的评价函数主要包括以下几种：

（1）距离

通过计算同一特征子集下同类样本与异类样本之间的距离来判定特征子集的好坏，若特征子集越好，则同类样本之间的距离越小。常用的判定距离主要由欧氏距离、马氏距离、闵可夫斯基距离等。

（2）相关性

通过计算特征子集与分类结果的相关性来判定特征子集的好坏，若特征子集越好，则与分类的相关性越高。常用的相关性判定系数主要有皮尔逊相关系数、余弦相似度、

杰卡德相似系数等。

2.6 本章小结

本章对文中涉及到的相关理论基础进行了介绍分析，包括 Spark 生态系统、Spark 相关技术，启发式算法相关理论以及随机森林算法的相关理论，最后对特征选择算法进行了简要阐述，为后文的相关研究奠定了理论基础。

3 Spark 负载均衡优化研究

Spark 负载均衡的目标是尽量使得 Spark 集群中各个计算节点任务负载相对均衡，使得计算资源能够被有效利用。Spark 集群的任务调度策略对于集群负载均衡具有重要影响，然而 Spark 集群在进行任务调度时并未结合各节点的实际情况，因此容易导致集群中节点负载不均衡。基于此，本章利用启发式算法寻求最优解的能力对任务调度策略进行了相关优化研究，通过改进任务调度策略，来达到平衡各节点负载，实现 Spark 负载均衡优化的目的。

3.1 Spark 负载均衡问题分析

集群负载均衡对于确保 Spark 集群的资源有效利用十分重要，Spark 中各节点任务分配的不均衡是导致集群负载不均衡的重要原因之一。在 Spark 进行任务调度时，遵循的是基于数据本地性的延迟调度任务分配原则[46]，即按照数据本地性从高到低进行任务调度，优先考虑将任务分配至数据本地性最优的节点，若此时计算节点中计算资源被占用，则进行等待，若任务在等待了一定时间后仍然无法执行，则降低数据本地性级别，并进行判定，若任务能够被执行则在该节点进行执行，若一定时间内仍不能被执行则继续降低数据本地性级别直至任务被允许执行。在 Spark 中，数据本地性从高到低分为五个等级，如表 3-1 所示。

表 3-1 数据本地性级别

数据本地性级别	级别说明
Process_Local	进程本地化，即数据与任务在同一进程中
Node_Local	节点本地化，即数据与任务在同一节点中
No_Pref	数据在其他数据源中读取
Rack_Local	机架本地化，即数据与任务在同一机架中
Any	数据在更远的节点中

但是此种分配策略在使用时，忽略了各节点的实际情况，易出现节点任务分配不均衡情况：即当前节点资源较丰富，计算能力较高，但是任务分配较少，处于低负荷状态，

负载较低；而节点剩余资源较少，计算能力较弱，处于高负荷状态下的节点，被分配了较多任务，处于高负荷状态，造成负载不均衡情况。例如运行一个作业，如果数据全都堆积在某一台节点上，那将只会有这台机器在长期执行任务，集群中的其他机器则会处于等待状态（等待数据本地性降级）而不执行任务，造成集群的负载失衡，同时也会造成大量的资源浪费，严重影响作业执行效率。而且只有单一节点进行计算，会大大增加整个作业的处理时间。因此，在进行任务调度时，结合节点当前实际情况进行任务分配，对于平衡节点负载，提高作业执行效率具有重要意义。

3.2 负载均衡优化策略分析

由于任务调度时遵循数据本地性优先的调度方式，忽略了各个计算节点的实际情况，因而引发负载不均衡，本文从任务调度的角度对 Spark 集群负载均衡进行优化，通过引入启发式算法改进任务调度策略，将待执行任务合理分配到 Spark 集群中的计算节点，来实现各个计算节点之间任务负载均衡，达到 Spark 负载均衡优化的目的。

任务调度问题描述如下：利用任务调度方案，将 n 个相互独立的 Task 分配给 m 个计算节点执行，如图 3-1 所示。而分配的目标就是在将这些任务分配到执行节点后，各个节点的任务负载相对均衡。通过平衡计算过程中各节点的任务负载，实现整个集群的负载均衡。

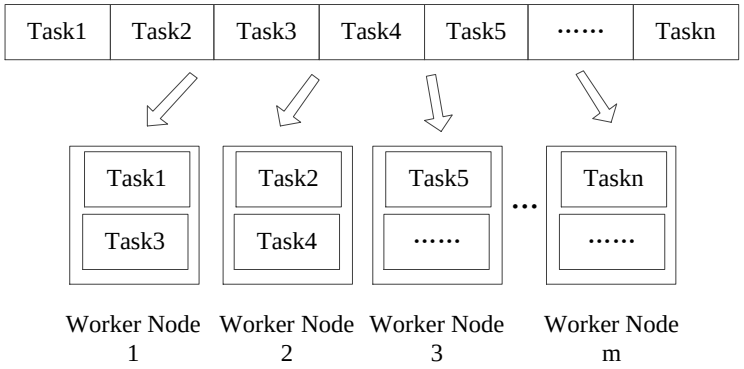

图 3-1 任务分配问题描述

3.3 相关理论基础

3.3.1 蚁群算法

1. 蚁群算法数学模型

蚁群算法最初被用来解决 TSP 问题[47]，在进行计算时，不同蚂蚁的不同行走路径表示了问题的多种可行解，而这些路径集合组成问题可行解集合，路径越短，则该路径信息素浓度越高，该路径上蚂蚁越多，经过多次的迭代，这种正反馈作用会使得蚂蚁向最优路径集中，此时即得到了问题的最优解。

具体数学模型为：假设城市数量为 n，蚁群蚂蚁数量为 m，城市之间的距离记为d_{ij}（i,j=1,2,3...,n），t 时刻 i，j 两座城市之间的路径信息素浓度用$\tau_{ij}(t)$表示，不同城市之间每条路径上的初始信息素浓度为τ_0。蚂蚁编号为 k（k=1,2,3,...,m），则在 t 时刻蚂蚁 k 由城市 i 像城市 j 转移的概率如式（3-1）所示。

$$P_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]\beta}{\sum\limits_{s\in allowed}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]\beta}, s\in allowed\\ 0, s\notin allowed\end{cases} \tag{3-1}$$

在式（3-1）中，*allowed* 表示可供蚂蚁选择的下一个城市集合，α 为信息素启发因子，β 为期望启发式因子，η 为启发函数。

在蚁群算法中，为了避免启发信息被覆盖的情况，算法模拟了现实生活中的信息素挥发机制引入了信息素更新思想[48]，即较少被选择路径的信息素会降低，从而使得大多数蚂蚁不再选择该路径。当所有蚂蚁完成一周循环后，信息素的更新如式（3-2）所示。

$$\tau_{ij}\left(t+1\right)=\left(1-\rho\right)\bullet\tau_{ij}\left(t\right)+\Delta\tau_{ij},0<\rho<1 \tag{3-2}$$

在上式中，$\Delta\tau_{ij}$为路径(i,j)上信息素增加的总浓度，其计算方式如式（3-3）所示。

$$\Delta\tau_{ij}=\sum_{k=1}^{m}\Delta\tau_{ij}^{k} \tag{3-3}$$

在式（3-3）中，$\Delta\tau_{ij}^{k}$为第 k 只蚂蚁在路径(i,j)上的释放信息素而增加的信息素浓度，其计算方式如式（3-4）所示，其中 Q 为信息素常数，L_k则是第 k 只蚂蚁单次循环中的路径距离总长。

$$\Delta\tau_{ij}^{k}=\begin{cases}\dfrac{Q}{L_k}, & \text{蚂蚁经过路径}(i,\ j)\\ 0, & \text{蚂蚁未经过路径}(i,\ j)\end{cases} \tag{3-4}$$

2. 蚁群算法优缺点

蚁群算法其优点主要有：

（1）算法拥有较强的适应性，且鲁棒性较好。

（2）蚁群算法并行性较好，且在数据规模较大时的处理效果较为明显。

（3）蚁群算法的正反馈机制使得算法可以尽快向最优解靠近。

当然，蚁群算法也有一定的局限性，具体表现在：

（1）算法对于各项参数较为敏感。

（2）算法机制使得蚁群算法容易陷入局部最优解的情况。

3.3.2 模拟退火算法

1. 模拟退火算法思想

模拟退火算法起源于冶炼时的金属降温过程，即先将金属加温，使其内部粒子剧烈变化，随后对其进行冷却，使得其内部粒子趋于稳定。而在将该思想应用于最优解问题时，用目标函数模拟粒子的变化程度，同时用参数代表温度，即是模拟退火算法的主要思想。

模拟退火算法的数学模型为：当初始温度为 t 时，对初始解 i 及随机产生的新解 j，进行目标函数值计算，若新解 j 的目标函数值优于 i，则用 j 替换 i，并对 j 进行此过程；若产生的新解 j 的目标函数值比 i 差，则以一定概率进行解的替换，而且该概率随着温度参数值 t 的降低而降低。经过多次上述迭代，最终算法会得到一个稳定解或者温度参数值降低至预设阈值，此时的解即为模拟退火算法所得到的近似最优解。模拟退火算法主要流程为：

Step 1：初始化参数，并随机生成初始解

Step 2：将初始解扰动置换产生新解，并判断新解是否符合算法要求

Step 3：若新解符合要求，则接受新解，若新解不符合要求，则按照 Metropolis 规则判断是否接受新解

Step 4：判断是否达到最大迭代次数，若达到最大迭代次数则算法结束，若未达到最大迭代次数，则返回步骤 2。

2. Metropolis 准则

在模拟退火算法当中，对经过对初始解进行随机扰动后形成的新解是否接受十分重要。模拟退火算法获得初始解后，通过置换规则对当前解进行随机节点交换，若生成的新解适应度函数值低于原来解，则接受新解，否则根据新解接受概率接受新解，此即为 Metropolis 准则[49]。

在 Metropolis 准则中，先对新解的效果进行计算，计算方式如式（3-5）所示。

$$\Delta T=target_{new}-target_{old} \tag{3-5}$$

在式（3-5）中，$target_{new}$ 表示通过置换规则得到的新的生成解的适应度函数值。

随后进行新解接受概率的计算，P_{allow} 表示新的解被接受的概率，即当新解适应度函数低于原来解时则接受新解，否则按照概率接受新解，其计算方式如式（3-6）所示。

$$p_{allow}=\begin{cases} e^{-\frac{\Delta T}{T}}, \Delta T>0 \\ 1, \Delta T<0 \end{cases} \tag{3-6}$$

3. 模拟退火算法优缺点

模拟退火算法在进行复杂区域的搜索时具有较好的效果，且因为其求解过程使其隐含了可并行性的特点，此外由于模拟退火算法搜寻过程是随机的，因此增强了算法的灵活性，使得算法可以较好地搜索到全局最优解。

3.4 基于蚁群-模拟退火的 Spark 负载均衡优化

3.4.1 蚁群-模拟退火融合算法

由于蚁群算法容易陷入局部最优解，因此针对蚁群算法存在的问题，本文引入模拟退火算法的随机扰动原则，当蚁群算法完成单次群体计算后产生当前次群体的最优解时，对该解进行局部随机扰动，形成新解，同时计算新解的适应度函数值，若新解比扰动之前的解效果更好，则接受新解为当前次群体的最优解，并进行全局信息素更新，进行优秀路径信息素强化，以加快算法的整体收敛速度，同时可以避免蚁群算法陷入局部最优解。

融合算法流程如图 3-2 所示。

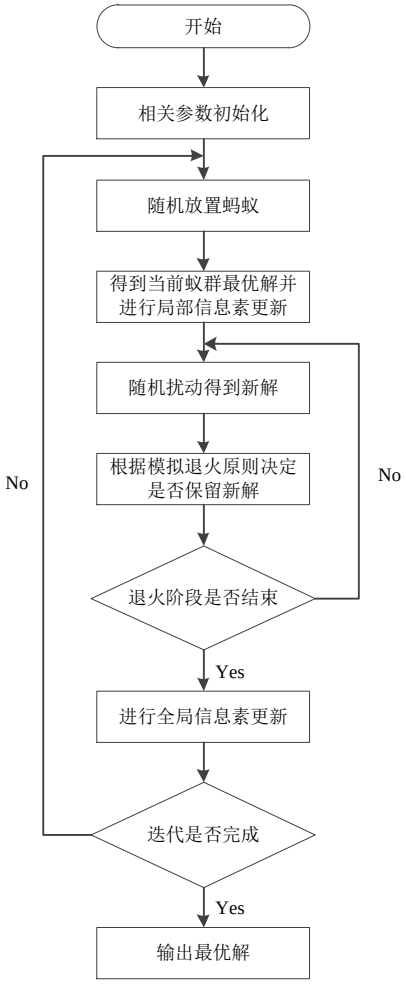

图 3-2 算法流程图

具体步骤如下：

Step 1.先将算法相关参数初始化，包括蚁群规模，各项参数权值，信息素挥发因子以及模拟退火初始温度等。

Step 2.将蚂蚁分布到不同节点上，并根据目标函数生成当次迭代最优解，并进行局

部信息素更新。

Step 3.根据置换规则构造新解，并根据 Metropolis 准则决定是否接受新解。

Step 4.根据生成的解进行全局信息素更新。

Step 5.判断是否达到迭代最大次数，若满足条件则算法结束，否则继续迭代。

3.4.2 负载评价指标

对于本文要解决的负载均衡优化问题，本文使用任务节点 CPU 使用情况 *Cm* 与节点任务量作为衡量当前节点负载情况的标准，计算方式如式（3-7）所示。

$$Load_i=\mu_1 \bullet Cm_i+\mu_2 \bullet \frac{taskpoint_i}{\sum_{i=1}^{m} taskpoint_i} \tag{3-7}$$

在式（3-7）中，$Load_i$ 表示第 i 个节点的负载情况衡量指标，$taskpoint_i$ 代表分配至该节点的任务量，μ_1 与 μ_2 为可调整参数。

所有任务分配至节点后节点实现负载均衡的期望值 E_{hope} 的计算方式如式（3-8）所示。

$$E_{hope}=\frac{\sum_{i=1}^{m} Load_i}{m} \tag{3-8}$$

节点负载偏差度 *Loaderror* 计算方式如式（3-9）所示。

$$Loaderror=\sqrt{\frac{\sum_{i=1}^{m}(Load_i-E_{hope})^2}{m}} \tag{3-9}$$

在式（3-9）中，*Loaderror* 表示当前任务分配方式与负载均衡期望值之间的偏差程度，即 *Loaderror* 值越小，则该分配方式的负载偏差程度越低，集群负载均衡程度越高。

3.4.3 信息素更新机制

本文采用局部更新与全局更新相结合的方式更新各个节点的信息素，即先对每只蚂蚁的节点信息素进行局部更新，待全部蚂蚁完成单次寻优后，再对较优节点增加优质信息素，从而实现全局更新，加快收敛速度。

基于本文要解决的负载均衡问题，节点负载越大，则说明节点当前资源占用率较高，可用资源较少，节点的计算能力较弱，故蚂蚁 k 信息素增量计算方式如式（3-10）所示。

$$\Delta\tau_{ij}^{k}=\frac{Q}{Load_i} \tag{3-10}$$

待所有蚂蚁完成单次寻优之后，由于蚂蚁在释放信息素的同时，各个节点的信息素强度也在通过挥发等方式逐渐减少，故全局信息素更新如式（3-11）所示。

$$\begin{cases}\tau_{ij}(t+1)=(1-\rho)\bullet\tau_{ij}(t)+\Delta\tau_{ij}+\Delta\tau',0<\rho<1\\ \Delta\tau_{ij}=\sum_{k=1}^{m}\Delta\tau_{ij}^{k}\\ \Delta\tau'=\begin{cases}\varphi\cdot\Delta\tau_{ij}, & \text{（最优解）}\\ 0, & \text{（非最优解）}\end{cases}\\ \varphi=\dfrac{\Delta Load_i}{\sum_{i=1}^{m}\Delta Load_i}\end{cases} \tag{3-11}$$

在蚂蚁完成一周循环后，进行解的随机扰动，并按照 Metropolis 准则决定是否保留新解，并在当前次迭代结束后对最优解进行额外信息素增加以凸显该解。其中 $\Delta\tau'$ 为优质解信息素增加量，用于缩短求解时间。

3.4.4 适应度函数设计

本章中适应度函数包含两部分，一部分为负载偏差度，另一部分为任务的完成时间，适应度函数设计如式（3-12）所示，其中 α 与 β 为可调整参数。

$$target=\alpha\bullet Loaderror+\beta\bullet Task_{time} \tag{3-12}$$

在式（3-12）中，$Task_{time}$ 即为当前分配方式下任务完成总时间，是所有节点中各节点完成该节点内所有任务所需时间的最大值。计算方式如式（3-13）所示，其中 V_{Pi} 表示节点 i 的计算速度。

$$Task_{time} = \max_{i=1}^{m}\{\frac{taskpo\,\mathrm{int}_i}{Vp_i}\} \tag{3-13}$$

3.4.5 Metropolis 接受函数

基于本文要解决的问题，将优化目标与接受函数相结合，以进行任务调度策略的新解接受，因此接受函数定义如式（3-14）所示。

$$\begin{cases} \Delta T = Loaderror_{new} - Loaderror_{old} \\ p_{allow} = \begin{cases} e^{-\frac{\Delta T}{T}}, \Delta T > 0 \\ 1, \Delta T < 0 \end{cases} \end{cases} \tag{3-14}$$

3.4.6 蚁群-模拟退火任务分配策略

随后本文将蚁群与模拟退火算法的融合算法应用到 Spark 任务分配策略的优化当中中，具体步骤如下：

Step 1：用户向集群提交 Spark 作业，并且初始化融合算法参数值；

Step 2：集群资源管理器获取各个节点的具体情况；

Step 3：在获得各个节点目前的运行状态、健康情况和负载信息后，进行路径选择，并进行局部信息素更新；

Step 4：当蚁群完成计算后，对于当前的局部最优解进行扰动，并根据 Metropolis 准则决定是否形成新解；

Step 5：完成单轮计算后，对于优质解增加额外信息素，完成全局更新；

Step 6：当算法达到终止标准时输出全局最优位置，该解对应的任务分配序列即为本次的最优调度方案，否则继续进行迭代计算；

Step 7：集群根据得到的任务分配方案进行任务分配。

3.5 实验验证与分析

本文实验分为两部分：第一部分在模拟环境下对算法性能进行对比验证；第二部分则是将 Spark 原生任务调度策略与本文提出的基于改进任务调度策略进行对比验证，验证基于改进任务调度策略的负载均衡优化效果。

3.5.1 模拟验证实验

本章在模拟环境下分别采用蚁群算法与蚁群模拟退火融合算法进行任务调度，并进行了适应度函数值与任务预期总体完成时间的比较，实验结果如图 3-3、图 3-4 所示。

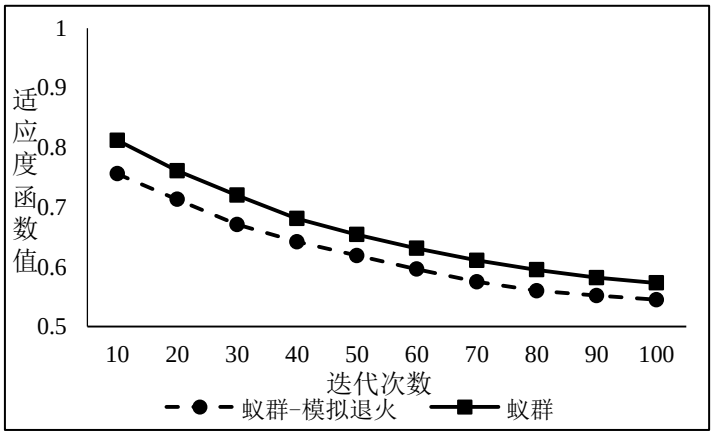

图 3-3 适应度函数值比较

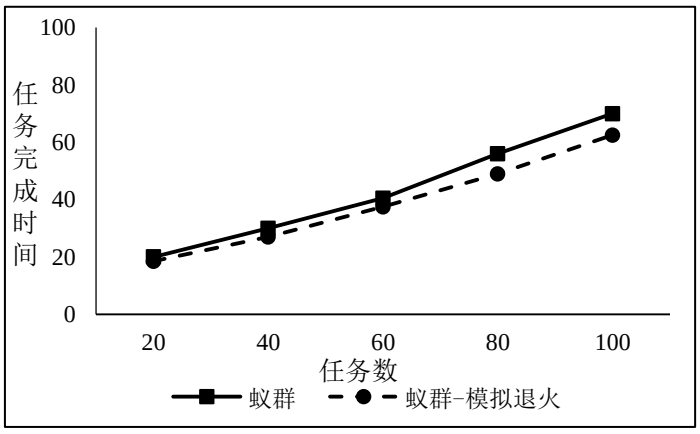

图 3-4 任务完成时间比较

由图 3-3 可知，在使用蚁群模拟退火融合算法进行任务调度时，适应度函数值明显优于蚁群算法的适应度函数值。

同时，由图 3-4 可知采用蚁群模拟退火融合算法进行任务调度的任务完成时间明显优于蚁群算法进行任务调度得到的任务完成时间，因此蚁群模拟退火融合算法在解决基于任务调度的负载均衡优化问题时具有更好的效果。

3.5.2 集群实验验证

在验证时，首先需要进行 Spark 集群安装配置，Spark 集群安装配置过程如下：

1.Hadoop 安装配置

由于本章使用的 Spark 集群以 Hadoop 平台为基础构建，因此需要先进行 Hadoop 平台的配置，具体步骤如下：

a. 配置各节点的主机名、网络及 hosts 文件。

b. 在官网中下载 JDK 文件，并在集群中的每个节点均进行 JDK 的安装及环境变量配置。

c. 在官网中下载 Hadoop 安装包，并在各个节点均进行环境变量及文件系统的配置。

d. 所有节点配置完成后，启动 Hadoop 集群，建立工作目录，并对文件系统进行格式化。

2. Spark 安装配置

a. 下载 Scala 安装包并进行安装。

b. 本章使用 IDEA 进行 Spark 源码的阅读与修改，因此需要下载并安装 IDEA。

c. 在 IDEA 中修改 Spark 源码并使用 Maven 进行编译。

d. 生成 Spark 安装包后进行 Spark 集群的安装配置。本文所使用的 Spark 集群共 5 个节点，其中 1 个为主节点，其余 4 个为工作节点，集群的软件配置如表 3-2 所示。

表 3-2 Spark 集群软件配置

参数名称	参数配置
Spark 版本	2.0.1
Hadoop 版本	2.7.1
Scala 版本	2.11.8

随后本文在集群环境下使用不同大小的商品评论数据集对本文提出的改进任务调度策略进行实验验证，主要包括两方面：

（1）验证在负载偏差度方面的改进。

（2）验证在任务总体完成时间方面的改进。

实验结果比较：

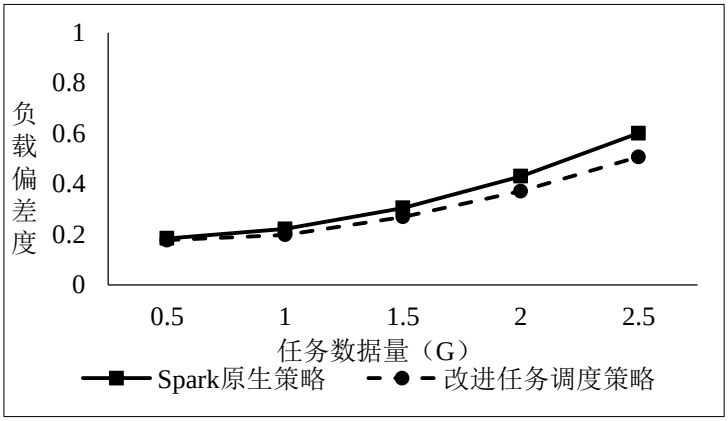

图 3-5 负载偏差度对比

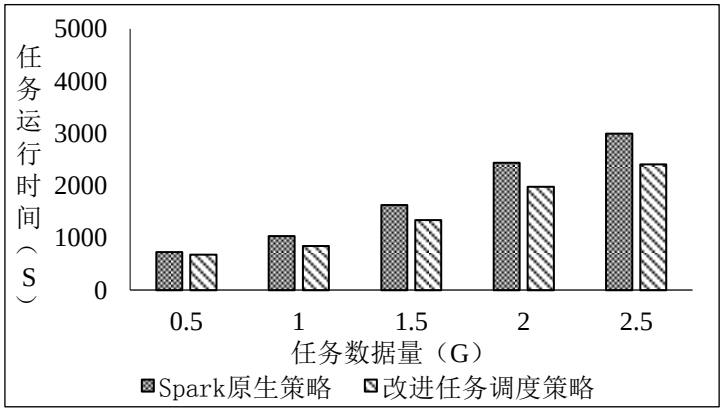

图 3-6 任务运行时间对比

由图 3-5 可知，随着任务数据量的增加，Spark 原生策略及本文提出的改进任务调度策略的负载偏差度均有所上升，但是本文提出的改进的任务调度策略负载偏差度上升较为缓慢，具有较好的负载均衡优化效果。

由图 3-6 可知，随着数据量的增加，本文提出的改进负载均衡策略在任务运行时间方面具有较好的改进效果。

综合对比图 3-5 和图 3-6 可知，与 Spark 原生任务调度策略相比，本文提出的基于蚁群模拟退火融合算法的改进任务调度策略不仅能够较好的实现 Spark 集群的负载均衡

优化，还在任务运行时间方面的具有较好的优化效果。

3.6 本章小结

Spark 是目前最为主流的分布式计算框架之一，Spark 计算性能的高低，影响着大数据处理分析的效率。针对 Spark 集群的负载不均衡问题，本文对其进行了研究分析，并提出了一种优化的 Spark 任务调度策略，该策略所实施的任务调度方案，对于 Spark 集群的负载均衡具有较为明显的优化效果，同时又缩短了任务的总体运行时间，提高了计算效率。

4 基于 Spark 的随机森林算法优化研究

随机森林算法是一种有效的数据挖掘算法，但是仍然无法避免在特征子空间生成时随机选择特征带来的冗余、噪音特征对于特征子空间有效性的影响，进而影响算法分类准确性。基于此，本章对于随机森林算法的特征子空间生成策略进行了研究，利用基于特征重要性选择特征用于生成特征子空间的方式改进了特征子空间生成策略，进而提升了算法的分类准确性，并基于 Spark 对优化后的随机森林算法进行了并行化实现。

4.1 随机森林算法问题分析

随机森林算法的优势主要是由于在训练过程中决策树训练子集生成的随机性以及决策树特征子空间的随机性，因此决策树的分类精度以及不同决策树之间的差异性，对于分类器性能的优劣具有重要影响。为了保证分类器的分类效果，必须要通过保证训练子集和特征子空间的有效性和多样性来确保决策树的分类强度以及不同决策树之间的差异性。

特征子空间的生成是随机森林算法中的重要步骤，同时随机森林算法的决策树强度及决策树之间的差异性也与特征子空间的生成有关。特征子空间越小，随机性越大，则决策树之间的差异性越大，但是会使得有效信息较少，降低分类正确率；反之，特征子空间越大，则有效信息越多，单棵决策树的分类效果越强，但是会降低决策树之间的差异性，影响算法的分类精度。

在数据中，通常包含有噪音特征及冗余特征，这些特征与类别的相关性较弱，同时包含的有效分类信息较少，而随机森林算法在特征子空间生成时，采用的是简单随机抽样的方式，该方式容易导致生成的特征子空间中含有较多的噪音及冗余特征，影响决策树的分类性能。

为了解决随机森林算法在特征选择中存在的问题，国内外一些研究人员提出了一系列的解决方法。Oshiro T M 等[50]证明了随机森林算法中不同特征子集对于分类准确率的影响；Robnik[51]则将信息增益与信息增益比进行结合来选择进行节点分裂的特征，但是该方法降低了算法的计算效率。Chen 等[52]在特征子集生成阶段将随机抽样方法进行了改进，使用信息增益进行特征选择，但是算法的稳定性扔有待提高。刘凯等[53]使用卡方

检验用于特征重要性的计算，并使用 Group Lasso 改进了特征选择过程，但是该算法由于增加了计算量，因此降低了算法的运行效率。

综上所述，为了保证随机森林算法的分类准确率，同时又能保证随机森林算法具有一定的稳定性，本文通过计算特征重要性，对特征进行区分，进而基于特征重要性来生成特征子空间。这种方式既能够提升决策树的分类强度，进而提升算法整体的分类效果，同时在特征抽取时抽取分类效果较弱的特征加入，能够保证不同的决策树之间的差异性，保证算法的分类能力。

4.2 相关理论基础

Relief 算法是一种过滤式特征选择算法，适用于两类样本的问题。Relief 算法的基本思想是给特征集中每一个特征赋予相应的权值，并迭代更新权值，使得对于分类结果影响较大的特征获得较大权值，而对于分类结果影响较小的特征获得较小的权值[54]。

Relief 算法通过对比特征在同类近邻点和异类近邻点间的差异来衡量特征对于分类的影响。如果某个特征在同类样本间的差异较小，在异类样本间的差异较大，则表明该特征的区分能力较强[55]。反之则说明该特征的区分能力较弱。特征间的差异用距离来表示，其中距离计算方式如下：

当特征为离散型特征时，距离计算方式如式（4-1）所示。

$$Diff(X_1^A, X_2^A) = \begin{cases} 0, X_1^A = X_2^A \\ 1, X_1^A \neq X_2^A \end{cases} \tag{4-1}$$

当特征为连续型特征时，距离计算方式如式（4-2）所示。

$$Diff(X_1^A, R_2^A) = \left| X_1^A - X_2^A \right| \tag{4-2}$$

在上式中，X_1 和 X_2 表示两个样本，X_1^A 和 X_2^A 分别表示两个样本在特征 A 上的特征值。

由于 Relief 算法仅适用于二分类问题，面对高维多分类数据时则较为乏力，因此 Kononenko[56]对 Relief 算法进行了扩展，设计出了 Relief-F 算法。Relief-F 算法在面对高维多分类数据时具有较好的效果，本章也正是引用了该算法的思想，其计算思想如下。

在数据集 S 中随机抽取一个属于类别 C 的样本 X_i，首先在 X_i 的同类样本中寻找 k 个最近邻样本记为 H_j（j=1,2,3,…,k），然后在与 X_i 不同类别样本集中各抽出 k 个近邻样本记为 F_{dj}(j=1,2,3,…,k)，其中 d 表示与 X_i 所属类 C 的某个不同类，并对 X_i 与 H_j 及 F_{dj} 的距离进行计算，将此过程重复进行 m 次，即可得到特征 A 的特征重要性。

特征 A 的重要性计算方式如式（4-3）所示。

$$W(A) = w(A) - D_{H_j} - D_{Fd_j} \tag{4-3}$$

其中 D_{H_j} 与 D_{Fd_j} 计算方式分别如式（4-4）、式（4-5）所示

$$D_{H_j} = \frac{\sum_{j=1}^{k} Diff(X_i^A, H_j^A)}{mk} \tag{4-4}$$

$$D_{Fd_j} = \frac{\sum_{d \neq C}\left[\frac{P(d)}{1-P(C)}\sum_{j=1}^{k} Diff(X_i^A, Fd_j^A)\right]}{mk} \tag{4-5}$$

在式（4-5）中，X_i^A 表示样本 X_i 在特征 A 的取值，P_d 为 d 类样本在数据集中所占比例，计算方式如式（4-6）所示，。

$$P(d) = \frac{num(d)}{num(S)} \tag{4-6}$$

4.3 基于特征重要性的随机森林算法

在随机森林算法中，由于算法在进行特征选取时采用随机选取的策略，因此很难避免一些冗余特征影响到随机森林算法的准确率和稳定性。因此本文引入 Relief-F 算法对于特征重要性进行计算以进行特征区分。

本文提出基于特征重要性的特征子空间的生成策略主要改进思路为：首先利用 Relief-F 算法计算特征与类别之间的关系，对特征重要性进行度量，即权重越高的特征，重要性越高，在分类时的类别相关度就越高，反之，权重越低的特征，重要性越低，在

分类时的类别相关度越低。在计算所有特征的重要性之后，得到特征重要性较高的一部分特征为强相关特征 *NS*，余下的特征为弱相关特征 *NW*。如图 4-1 所示。

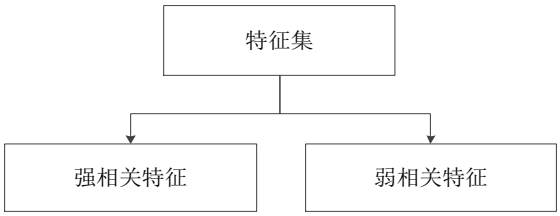

图 4-1 强弱相关特征区分

区分强弱相关特征之后，在决策树训练的特征抽取阶段，不采用传统随机森林算法的随机抽取特征的方式，而采用分层抽取特征形成特征子空间的生成方式，以此保证决策树的有效性和差异性。

改进的特征子空间生成流程如图 4-2 所示。

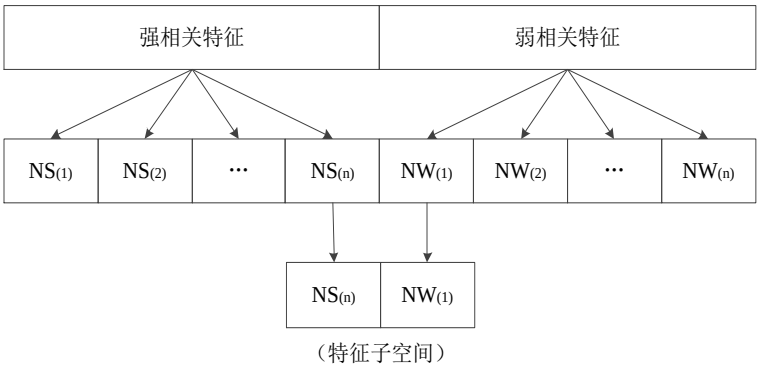

图 4-2 特征子空间生成流程

在进行分层抽取时，强弱相关特征的比例与强弱相关特征的重要性占比相关，具体表示为：假设在特征子空间中包含 *Sub* 个特征，则在特征子空间中强相关特征的数量 Num_{NS} 的计算如式（4-7）所示：

$$Num_{NS} = Sub \bullet S_{NS} \tag{4-7}$$

其中 S_{NS} 为强相关特征重要性在所有特征重要性中所占比例，计算方式如式（4-8）所示。

$$S_{NS}=\frac{\sum_{A\in NS}W(A)}{\sum_{A\in NS\cup NW}W(A)} \tag{4-8}$$

在构成特征子空间的过程中，由于强相关特征的特征重要性较高，因此通过保证特征子空间中强相关特征的比例，可以确保特征子空间的有效性，而弱相关特征的选取，使得不同决策树之间抽取到的特征不尽相同，以此保证不同决策树之间的差异性。

在生成特征子空间之后，即可进行决策树训练，进而集成为随机森林模型。

优化随机森林算法伪代码如表 4-1 所示。

表 4-1 优化随机森林算法伪代码

算法伪代码：
输入：训练集 D　决策树个数 NumTree　　单棵决策树最大深度 Depth 输出：随机森林模型 1. For (f from 1 to 特征总个数 m) 计算特征重要性 End 2. 按照特征重要性将特征分为强相关特征与弱相关特征 3. For(i from 1 to NumTree) 1） 对训练集 D 进行抽样获得抽样样本用于决策树训练 2） 对得到的强相关特征与弱相关特征分别进行抽取合成特征子空间 3） 训练决策树 End 4. 集成决策树为随机森林模型

优化随机森林算法的具体步骤如下：

Step1：随机有放回抽取 K 个样本子集。

Step2：根据 Relief-F 算法计算得到各个特征重要性。

Step3：根据本文提出的特征子空间生成策略进行特征子空间选取。

Step4：进行决策树训练，决策树训练过程中的节点分裂属性的选择以基尼系数为选

择方式。基尼系数计算方式如式（4-9）所示。

$$Gini(D,A)=\frac{|D_1|}{|D|}Gini(D_1)+\frac{|D_2|}{|D|}Gini(D_2) \tag{4-9}$$

其中

$$Gini=\sum_{i=1}^{k}P_i(1-P_i)=1-\sum_{i=1}^{k}P_i^2 \tag{4-10}$$

Step5：将 Step2-4 重复进行直至所有决策树训练完毕，得到随机森林模型

Step6：投票得到最终输出结果。

4.4 基于 Spark 的改进随机森林算法并行化设计

4.4.1 算法整体并行化设计

在使用 Spark 进行数据分类时，需要先将待分类数据转换为 Spark 集群中使用的 RDD，并使用相关算子完成算法整体并行化，具体流程如图 4-3 所示。

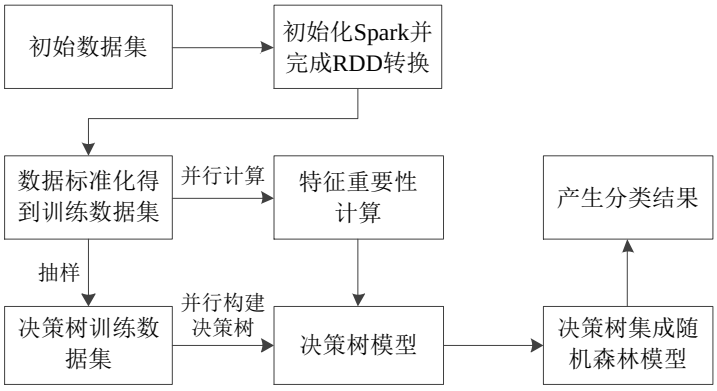

图 4-3 算法整体并行化设计

4.4.2 特征重要性计算并行化设计

在进行特征重要性计算的过程中，由于每个抽取到的样本均需要对其进行多次计算并进行权重更新，因此可对特征重要性计算过程进行并行化设计如下：

将样本的距离计算任务分配给计算节点，并将计算得到的结果存储在 Spark RDD 中，整个数据集则作为广播变量发送到节点，方便节点对其进行查找。该部分的主要设计思想包含三个步骤：

Step 1. 抽取 K 个样本并初始化特征权重

Step 2. 利用 map 操作进行特征重要性的并行计算

Step 3. 利用 reduce 操作进行特征的重要性整合

具体过程如图 4-4 所示。

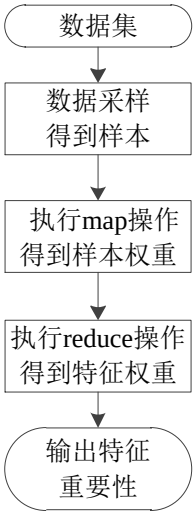

图 4-4 特征重要性计算并行化过程

4.4.3 随机森林模型建模并行化设计

多棵决策树的生成在随机森林算法的运行中占了很大的比例。但是随机森林算法在进行决策树训练的过程中，每一棵决策树的生成都是相互独立地抽取数据样本，抽取特征分裂节点，生成决策树，因此在随机森林建树阶段可以将多棵决策树的生成并行化。

主要设计思想：

（1）假设需生成 K 棵分类回归树，则抽取 K 个样本子集作为决策树训练集。

（2）利用数据集并行化训练决策树。

（3）将决策树集成为随机森林。

具体并行化过程如图 4-5 所示，设计步骤如下：

Step 1：假设需要生成的决策树数量为 K，则通过 bootstrap sample 方法抽取 K 个大小相同的样本子集作为决策树训练集。

Step 2：通过 map 操作在训练集中并行训练决策树。

Step 3：将决策树构建为随机森林。

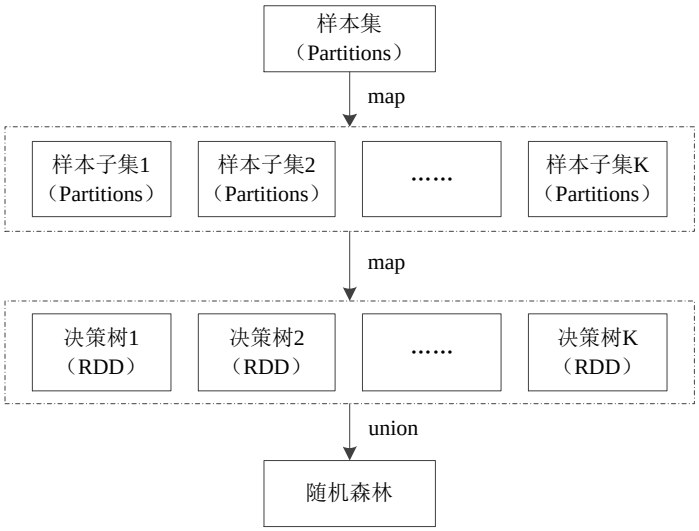

图 4-5 随机森林建模并行化流程

4.5 实验验证与应用

4.5.1 算法改进实验验证与分析

在分类器完成相关的任务后，对于分类效果进行评价是实验的重要环节。在分类效果评估中，混淆矩阵是一种十分重要的方式。以二分类为例，混淆矩阵如表 4-2 所示。

表 4-2 混淆矩阵

预测 \ 真实	0	1
0	TP	FP
1	FN	TN

如上表所示，在混淆矩阵中，TP 表示原本属于 0 类且被分类器正确划分为 0 类的样本数，FP 表示原本属于 1 类但是被分类器错误划分为 0 类的样本数，FN 表示原本属

于 0 类但是被分类器错误划分为 1 类的样本数，TN 表示原本属于 1 类且被分类器正确划分为 1 类的样本数。

对于分类器的评价评价指标主要由以下几种，具体计算方式如下：

（1）正确率。正确率是在进行分类器评估时的一个常用指标，具体计算方式如式（4-11）所示。

$$accuracy=\frac{TP+TN}{TP+FP+FN+TN} \tag{4-11}$$

（2）错误率。错误率代表分类结果中被错误分类的比例，具体计算方式如式（4-12）所示。

$$error=\frac{FP+FN}{TP+FP+FN+TN} \tag{4-12}$$

（3）召回率。召回率代表 0 类样本被正确分类的比例，具体计算方式如式（4-13）所示。

$$recall=\frac{TP}{TP+FN} \tag{4-13}$$

在本章中，选择分类准确率作为改进算法的评估指标。

本文使用 UCI 中的数据集进行对比验证，各个数据集特征维数如表 4-3 所示。

表 4-3 数据集特征维数情况

数据集序号	数据集名称	数据集特征维数
数据集 1	Wine	13
数据集 2	Breast	32
数据集 3	Sonar	60
数据集 4	Musk	168

随机森林算法改进前后分类准确率对比如图 4-6 所示。

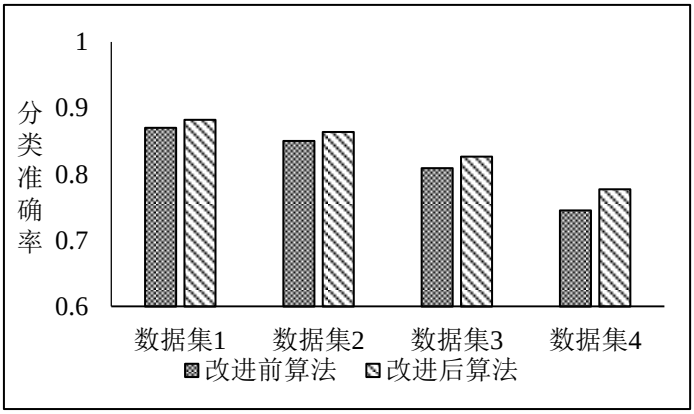

图 4-6 改进前后准确率对比

如图 4-6 所示，实验结果表明，本文提出的优化后的随机森林算法相比较于优化前的随机森林算法，算法分类准确率有较为明显的提升。同时，随着数据集特征维数的增加，本文提出的改进算法分类准确率提升效果更加明显。因此可以看出，本文提出的优化后的随机森林算法在特征维数较高的数据集上具有更好的效果。

随后本文在 Spark 集群环境下进行了算法加速效果验证。数据集为人造测试数据集，验证标准为加速比，加速比用于描述算法运行时间的减少效果。计算方式如式（4-14）所示。

$$\rho = \frac{t_1}{t_n} \tag{4-14}$$

在式（4-14）中，t_1 表示算法在单节点下的运行时间，t_n 表示算法在多个节点上并行化计算时的运行时间。算法的加速比变化如图 4-7 所示。

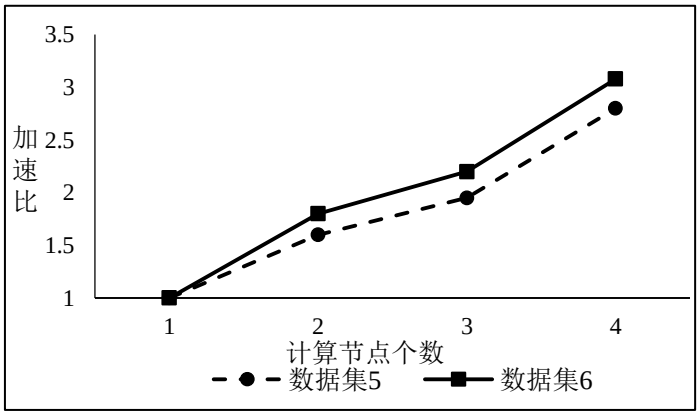

图 4-7 算法并行加速比曲线

由图 4-7 可以看出，随着 Spark 集群中工作节点的增加，算法加速比也随之增加。表明算法具有良好的并行化效果。且由于不同数据集的数据量数据集 6>数据集 5，说明算法在数据量增加的情况下具有更好的加速比。因此基于 Spark 进行优化算法的并行化能够有效地进行大数据分析处理。

4.5.2 在信用评估领域中的应用

随着经济的发展和人们消费观念的变化，使得信用消费被推向了新的高度。因此对用户的信用进行有效评估对于保证信贷业务的良好发展具有重要意义。本文将基于 Spark 的改进后的随机森林算法用于信用评估领域。分别使用 GermanCreditData 数据集及 Default of credit card clients 数据集进行信用评估应用，结果如图 4-8 所示。

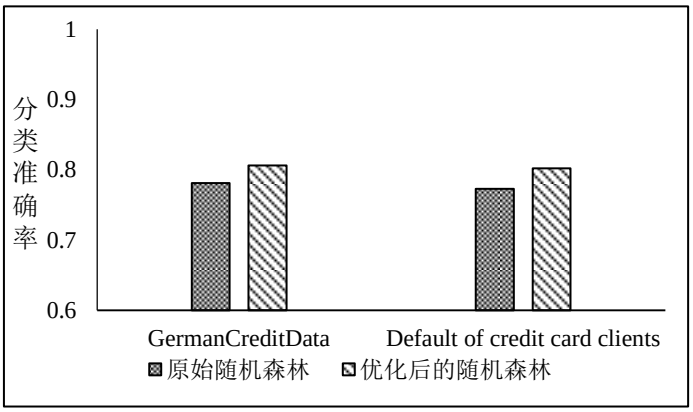

图 4-8 信用领域应用结果

通过图 4-8 实验结果可以看出，本文提出的优化后的随机森林算法相比于优化前的随机森林算法，在不同数据集中的分类准确率均有较为明显的提升。表明本文提出的基于 Spark 的随机森林算法在信用评估数据集中具有更好地分类能力，能够有效地应用于信用评估领域。

4.6 本章小结

本章主要介绍了传统随机森林算法在大数据时代所拥有的优势以及其存在的问题，针对随机森林算法所存在的问题，本文根据 Relief-F 算法计算特征的权值，用来区分强相关特征与弱相关特征，并使用分层抽样的方法获得特征子空间，用以提高随机森林算

法的分类效果，随后在 Spark 平台中对优化后的随机森林算法进行了并行化实现。最后通过分类性能实验对算法的性能进行比较和分析，证明了算法改进的有效性。

5 总结与展望

5.1 研究总结

本文针对于 Spark 负载均衡问题及基于 Spark 的随机森林算法进行了相关研究，具体包括以下研究：

（1）Spark 负载均衡优化研究

针对于 Spark 中的负载不均衡问题，本文提出了一种改进策略，该策略基于优化的任务分配策略实现负载均衡。针对于蚁群算法容易陷入局部最优解的问题，本文提出了蚁群模拟退火融合算法，融合算法将模拟退火算法中的随机扰动思想引入蚁群算法中，弥补蚁群算法的不足，在避免局部最优的同时缩短迭代时间。随后将蚁群模拟退火融合算法用于任务调度策略优化，提出了任务调度改进策略，并通过任务调度改进策略实现了对 Spark 集群的负载均衡优化。最后通过实验证明了本文提出策略的有效性。

（2）基于 Spark 的随机森林算法优化研究

针对于随机森林算法在进行特征子空间生成时采用简单随机抽样选择特征而无法区分噪音冗余特征的问题，本文对于随机森林算法中的特征子空间生成策略进行了改进，通过计算特征重要性对特征进行强弱相关特征区分，并采用分层抽取特征的方式保证特征的有效性及决策树之间的差异性，并基于 Spark 平台对优化的随机森林算法进行了并行化实现，最后通过实验证明了算法改进的有效性。

5.2 研究展望

本文对 Spark 负载均衡和基于 Spark 的随机森林算法进行了研究，提高了 Spark 集群在执行任务时的负载均衡程度以及随机森林算法的分类效果。但本文的研究在一定程度上也存在不足，在未来仍需进行深入研究，具体总结如下：

（1）Spark 负载均衡问题

1）本文在进行相关研究时，关于相关参数的设置，部分参数仍需采用经验值，因此下一步的研究方向是如何能够实现参数的智能化设定，以通过智能化设置参数来实现对集群负载均衡的进一步优化，提升集群整体性能。

2）本文在对 Spark 集群的负载不均衡程度进行衡量时考虑的因素较少，在后续研究中应当将其他有可能影响集群负载情况的因素考虑在内，使计算结果更为准确。

（2）Spark 随机森林算法优化问题

1）本文对 Spark 随机森林算法优化问题只考虑到特征选择的问题，在以后的研究中还应考虑不平衡数据集情况下对于算法分类准确度的影响。

2）如何在保证算法分类准确率的前提下进一步缩短算法执行时间也是下一步研究的重点方向。

参考文献

[1] 大数据产业生态联盟.2019 中国大数据产业发展白皮书[EB/OL]. (2019-09-11) [2020-02-11]. https://max. book118. com/html/2019/0311/5324330124002020. shtm

[2] Huang Y , Yesha Y , Halem M , et al. YinMem: A distributed parallel indexed in-memory computation system for large scale data analytics[C]. 2016 IEEE International Conference on Big Data (Big Data), 2016:214-222.

[3] Akil B , Zhou Y , Rohm U . On the usability of Hadoop MapReduce, Apache Spark & Apache flink for data science[C]. 2017 IEEE International Conference on Big Data. 2017:303-310.

[4] 高彦杰. Spark 大数据处理技术、应用与性能优化[M]. 北京: 机械工业出版社, 2015: 65-77.

[5] 牛志华. 基于 Spark 分布式平台的随机森林分类算法研究[D]. 中国民航大学, 2017.

[6] Wu H C , Wu Y T . Evaluating credit rating prediction by using the KMV model and random forest[J]. Kybernetes, 2016, 45(10):1637-1651.

[7] Mori H , Umezawa Y . Credit risk evaluation in power market with random forest[C]. 2007 IEEE International Conference on Systems, Man and Cybernetics. 2007:3737-3742.

[8] Zhang X. , Yang Y , Zhou Z . A novel credit scoring model based on optimized random forest[C]. 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), 2018: 60-65.

[9] Ko B C , Gim J W , Nam J Y . Cell image classification based on ensemble features and random forest[J]. Electronics Letters, 2011, 47(11):638.

[10] Xu B , Ye Y , Nie L . An improved random forest classifier for image classification[C]. 2012 IEEE International Conference on Information and Automation . 2012:795 - 800.

[11] 彭徵, 王灵矫, 郭华. 基于随机森林的文本分类并行化[J]. 计算机科学, 2018, 45(12):148-152.

[12] Tang Z, Zhang X, Li K, et al. An intermediate data placement algorithm for load balancing in Spark computing environment[J]. Future Generation ComputerSystems, 2018, 78(1): 287-301.

[13] 李巧巧. 面向负载均衡的 Spark 任务划分与调度策略研究[D]. 湖南大学, 2017.

[14] 黄超杰. Spark 中的数据均衡分配算法研究[D]. 电子科技大学, 2018.

[15] Liu G, Zhu X, Wang J , et al. SP-Partitioner: A novel partition method to handle intermediate data

skew in spark streaming[J]. Future Generation Computer Systems, 2018, 86:1054-1063.

[16] Aljawarneh I M, Bellavista P, Corradi A, et al. Efficient spark-based framework for big geospatial data query processing and analysis[C]. Computers and Communications. 2017: 851-856.

[17]杨志伟, 郑烇, 王嵩, 等. 异构 Spark 集群下自适应任务调度策略[J]. 计算机工程, 2016, 42(1): 31-35.

[18] Liang Y, Tang Y, Zhu X, er al. Task Scheduling Strategy for Heterogeneous Spark Clusters[C]. the International Conference on Artificial Intelligence in China. 2019:131-138.

[19] Xu L, Butt A R, Lim S, et al. A Heterogeneity-Aware Task Scheduler for Spark[C]. 2018 IEEE International Conference on Cluster Computing (CLUSTER), 2018: 245-256.

[20] Du H Z, Zhang K, Huang S, OctopusKing: A TCT-Aware Task Scheduling on Spark Platform[C]. 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), 2019: 159-162.

[21] Fan K , Zhang D , Li H , et al. An Adaptive Feedback Load Balancing Algorithm in HDFS[C]. 2013 5th International Conference on Intelligent Networking and Collaborative Systems, 2013: 23-29.

[22] 刘党朋. 不均衡环境下面向 Hadoop 的负载均衡算法研究[D]. 北京邮电大学, 2015.

[23] 刘盼红. 大数据环境下 Hadoop 作业调度算法的研究[D]. 河北工程大学, 2015

[24] 冯亮亮. 基于 Hadoop 的 MapReduce 性能优化研究[D]. 南京邮电大学, 2017.

[25] 姜淼. Hadoop 云平台下调度算法的研究[D]. 吉林大学, 2012.

[26] 于磊春. Hadoop 集群中数据负载均衡优化及其平台应用研究[D]. 江苏大学, 2018.

[27] 李贞贵. 随机森林改进的若干研究[D]. 厦门大学, 2013.

[28]Robnik-Sikonja M. Improving random forests[C]. European Conference on Machine Learning. 2004, 3201: 359-370.

[29] Amaratunga D, Cabrera J, Lee Y S. Enriched random forests[J]. Bioinformatics, 2008, 24(18): 2010-2014.

[30] 马春来, 单洪, 马涛, 等. 随机森林改进算法在 LBS 用户社会关系推断中的应用[J]. 小型微型计算机系统, 2016, (12):2708-2712.

[31] Joaqu ń A, Carlos J, Javier G, et al. Increasing diversity in random forest learning algorithm via imprecise probabilities[C]. Expert Systems with Applications, 2018, 97: 228-243.

[32] 王雪. 面向高维不平衡数据的随机森林算法及其并行化研究[D]. 辽宁大学, 2016.

[33] 罗元帅. 基于随机森林和 Spark 的并行文本分类算法研究[D]. 西南交通大学, 2016.

[34] 王日升. 基于 Spark 的一种改进的随机森林算法研究[D]. 太原理工大学, 2017.

[35] Xu Y, Research and implementation of improved random forest algorithm based on Spark[C]. 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), 2017: 499-503.

[36] Man W S, Ji Y Y, Zhang Z Z, Image classification based on improved random forest algorithm[C]. 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), 2018: 346-350.

[37] 肖怀远. 基于 Spark 的高效用项集挖掘算法研究[D]. 河南大学, 2018.

[38] 刘春雷. 基于代价模型的 Spark SQL 查询优化研究[D]. 电子科技大学, 2016.

[39] 胡天宇. 基于 Spark 的随机森林算法优化与并行化研究[D]. 齐鲁工业大学, 2019.

[40] Yan Y L, Chen M, Sadiq S. Efficient Imbalanced Multimedia Concept Retrieval by Deep Learning on Spark Clusters[J]. International Journal of Multimedia Data Engineering & Management, 2017, 8(1):1-20.

[41] 李筱川. 基于 Spark 的情报大数据可视化分析[D]. 山东大学, 2017.

[42] 文馨, 陈能成, 肖长江. 基于 Spark GraphX 和社交网络大数据的用户影响力分析[J]. 计算机应用研究, 2018, 35(3):830-834.

[43] 郭豪. 双权重随机森林预测算法及其并行化研究[D]. 哈尔滨工业大学, 2017.

[44] 李如平. 数据挖掘中决策树分类算法的研究[J]. 东华理工大学学报（自然科学版）, 2010, (2):96-100.

[45] 魏正韬, 杨有龙, 白婧. 基于非平衡数据的随机森林分类算法改进[J]. 重庆大学学报, 2018, . 41(4):58-66.

[46] 刘思宇. 基于执行时间评估的 Spark 任务调度技术研究[D]. 北京工业大学, 2017.

[47] 王会颖. 蚁群算法及群体智能的应用研究[D]. 安徽大学, 2007.

[48]赵宝江, 李士勇, 金俊. 基于自适应路径选择和信息素更新的蚁群算法[J]. 计算机工程与应用, 2007, 43(3):12-15.

[49] Haario H , Tamminen S J . An Adaptive Metropolis Algorithm[J]. Bernoulli, 2001, 7(2):223-242.

[50] Oshiro T M , Perez P S , José Augusto Baranauskas. How Many Trees in a Random Forest?[J]. 2012.

[51] Robnik-Sikonja M. Improving Random Forests[C]. In Proceedings of the 15th European Conference on Machine Learning. Italy：Computer Science，2004：359-372.

[52] Chen J，Li K，Member S，et al. A Parallel Random Forest Algorithm for Big Data in a Spark Cloud Computing Environment[J]. IEEE Transactions on Parallel and Distributed Systems，2017，28(4)：919-933.

[53] 刘凯，郑山红，蒋权，et al. 基于随机森林的自适应特征选择算法[J]. 计算机技术与发展，2018(9)：101-104, 111.

[54] 黄晓娟. 面向特征选择的 Relief 算法研究[D]. 苏州大学，2018.

[55] 陈晓琳，姬波，叶阳东. 一种基于 ReliefF 特征加权的 R-NIC 算法[J]. 计算机工程，2015，41(4)：161-165.

[56] Marko Robnik-Ikonja，Igor Kononenko. Theoretical and Empirical Analysis of ReliefF and RReliefF[J]. Machine Learning，53(1-2)：23-69.

作者简历

姓名：	张占峰	民族：	汉族
性别：	男	出生日期：	1996.01.01
籍贯：	河北唐山	最后学历：	硕士研究生
专业：	计算机应用技术	毕业院校：	河北经贸大学

学习经历：

2013 年 9 月至 2017 年 6 月 河北经贸大学信息技术学院 网络工程专业 攻读学士学位

2017 年 9 月至 2020 年 6 月 河北经贸大学信息技术学院 计算机应用技术专业 攻读硕士学位

攻读学位期间取得的科研成果：

（一）发表的学术论文

1. 第一作者：MapReduce 框架下常用聚类算法比较研究[J].河北省科学院学报,2019,36(02):1-6.
2. 第二作者：云计算资源调度算法比较研究[J].河北省科学院学报,2018,35(04):12-17.
3. 第二作者（导师第一作者）：Research on Load Balancing Algorithm Optimization Based on Spark Platform[C].ICAIS2019:452-465.（EI 收录，Accession number：20193207287577）
4. 第三作者：A Dynamic Memory Allocation Optimization Mechanism Based on Spark. CMC-Computers, Materials & Continua, 61(2), 739－757.（SCI 收录，WOS:000510452900019）

（二）参与项目

1. 2019 年度河北省研究生创新资助项目《Spark 平台下数据倾斜优化研究》，编号：CXZZSS2019106，主持人。
2. 教育部教育技术基金项目《大数据分析处理技术及其在绿色金融领域的应用研究》，编号：2017A01020，参加人。
3. 河北省教育厅教学改革重点项目《基于新工科理念面向景观领域大数据人才培养研究与实践》，编号：2017GJJG083，参加人。
4. 河北省教育厅教学改革项目《基于案例教学的大数据卓越人才培养模式研究与实践》，编号：2018GJJG180，参加人。

致　谢

时光荏苒，如白驹过隙，转眼间就要硕士研究生毕业了。回想过去，在河北经贸大学学习的几年里，我收获良多，在此向给予我帮助的河北经贸大学的全体老师和同学表示最诚挚的谢意。

首先要感谢我的硕士生导师王素贞教授。王老师在学习和生活中均给予了我大力的帮助。在学习上，王老师治学严谨，为我制定了合理的学习计划。在毕业论文的完成过程中，从选题到研究方案的确定，再到实验的拟定以及论文的撰写过程中，王老师认真负责，从论文大纲到定稿的过程中，一遍又一遍地对论文进行审阅，指出存在的问题，严格把关，一丝不苟。在生活上，王老师给予了我细致的关怀，时刻引导我逐步形成正确的人生观、价值观，并引领我渐渐走向成熟。

感谢河北经贸大学提供的优美的校园环境，感谢信息技术学院提供良好的学习条件，使得我能够在知识的海洋中尽情遨游。同时非常感谢信息技术学院的各位老师在我学习期间给予的无私帮助以及细心教导，是老师们教会了我诚实做人，踏实做事。

最后非常感谢我的各位同学和家人，感谢师姐张艳飘、张璐在我研究生刚入学时期提供指导和帮助，感谢同门同学耿珊珊、师弟王文礼在学术上的交流和帮助，感谢同寝室的室友和实验室的同学朋友们，在他们的帮助下，我解决了在学习以及生活中遇到的诸多问题。感谢我的父母对我的照顾，鼓励，支持以及无私的关爱。

最后，再次向所有关心我的亲人、老师和同学们表示深深的谢意。在即将离校之际，祝愿父母、所有老师和同学们健康快乐！

河北经贸大学研究生学院监制