Spark 平台下的高效 Web 文本分类系统的研究

李 涛 刘 斌
(南京工业大学计算机科学与技术学院 江苏 南京 210009)

摘 要 针对 KNN 分类算法在面对海量 Web 文本处理情况时在单机上训练和测试效率低下的问题,提出基于 Hadoop 分布式平台以及 Spark 并行计算模型的无中间结果输出的改进型 Web 文本分类系统。同时为了充分利用 Spark 的迭代计算能力,在文本向量化阶段,在传统 TFIDF 文本特征加权算法的基础上充分考虑特征项在类内和类间的信息分布,提出一种改进的特征加权算法。实验结果表明,该文本分类系统结合 Spark 计算模型在提高文本预处理、文本向量化以及 KNN 文本分类算法的性能上有着优异的表现。

关键词 KNN TFIDF 文本分类 Hadoop Spark

中图分类号 TP391.1 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2016.11.008

RESEARCH ON EFFICIENT WEB TEXT CLASSIFICATION SYSTEM BASED ON SPARK

Li Tao Liu Bin
(*College of Computer Science and Technology, Nanjing Technology University, Nanjing 210009, Jiangsu, China*)

Abstract In order to solve the problem of low efficiency of KNN classification algorithm in training and test on a single computer when facing the situation of processing massive Web texts, we proposed an improved Web text classification system without intermediate result output, which is based on Hadoop distributed platform and Spark parallel computing model. Meanwhile, in order to take full advantage of the computing power of Spark in iterative computation, at the stage of text vectorisation and on the basis of the traditional text feature weighting algorithm of TFIDF, we made the full consideration on the information distribution of the feature items within class and between class and proposed an improved feature weighting algorithm. Experimental results showed that this Web text classification system, in combination with Spark computing model, has excellent performance in improving text preprocessing, text vectorisation and the performance of KNN text classification algorithm.

Keywords KNN TFIDF Text classification Hadoop Spark

0 引 言

随着大数据浪潮的到来,对海量信息的处理能力已经成为一个相当重要的课题。成熟的文本分类系统通常具有很高准确率,但 Web 文本信息的实时性特点同时也要求分类系统具有很高的分类效率。目前使用比较广泛的文本分类算法包括 K 临近算法[1]、朴素贝叶斯[2]、最大熵[3]、支持向量机(SVM)[4]、人工神经网络[5]、决策树[6]、粗糙集[7]等等。对上述文本分类方法的研究都集在中小规模的数据模型上,当面对大规模数据且对实时性要求较高的场景,传统分类系统显得无能为力。本文在 Spark 平台下实现了一种 Web 文本分类系统,该系统对解决大规模、实时条件下的文本分类具有重要的现实意义。

1 Spark

与 MapReduce 不同的是,Spark 的并行计算框架是基于内存的。Spark 其实就是 MapReduce 的替代方案,它可以兼容 HDFS 和 Hive 等分布式存储层,可以融入 Hadoop 生态系统。Spark 相比于 MapReduce 的优势有诸多的优势[8],特别适用于流式计算和机器学习,可以在一个工作流中无缝搭配这些计算范式,所以本文采用 Hadoop 和 Spark 作为海量 Web 文本分类的实现平台。

2 Web 文本分类系统

文本分类系统的工作流程是:首先对收集到的文本进行预处理,如图 1 所示。然后利用 VSM 模型[9]将文本使用向量来表示。然后可以利用成熟的分类算法进行分类,如 KNN 分类算法。最后一步是结果评价。图 1 给出系统整体结构图。

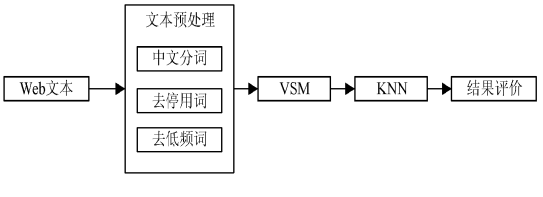

图 1 Web 文本分类系统结构图

收稿日期:2015-09-13。**李涛**,硕士生,主研领域:数据挖掘,文本分类。**刘斌**,博士。

2.1 文本预处理

要提高文本分类的效果,首先必须对文本进行预处理,其中的主要步骤包括:分词、去停用词和低频词等。在 Spark 集群上实现文本预处理,RDD 中的数据项就是 Web 文本中的每行内容。将文本内容进行分割去、除停用词,计算词频形成属性词典,这些分布式操作都是在 Worker 节点上完成的。预处理在 Spark 下的执行流程如图 2 所示。

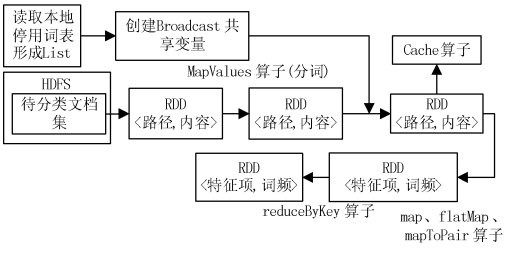

图 2 Spark 文本分类预处理结构图

2.2 文本向量化

(1) 经典的特征项加权算法 TFIDF

目前,在文本分类研究领域中,向量空间模型(VSM)是常用的形式。在该模型中,每篇文档被表示为一组特征向量,$d_j = \{(w_1, f_1), (w_2, f_2), \cdots, (w_n, f_n)\}$,其中 w_i 表示在文档 d_j 中出现的特征词,f_i 是 w_i 的权值,其中 i 的取值为 $1,2,\cdots,n$。其中特征词可在文本预处理阶段得到,特征词权重的计算利用经典的 TFIDF 加权算法。

TFIDF 算法由 Salton 提出[10],其中的 TF (Term Frequency) 表示某个特征项在某篇文档中出现的频率,计算方式如下:

$$TF_{wij} = \frac{n_{w_{ij}}}{\sum_k n_{w_{kj}}} \tag{1}$$

其中,w_{ij} 表示第 j 篇文档中的第 i 个特征项,$n_{w_{ij}}$ 表示该特征项在文档 d_j 中出现的次数,$\sum_k n_{w_{kj}}$ 表示文档 d_j 中所有特征项出现次数之和。

逆向文档频率 IDF (Inverse document frequency) 实现对词语重要性的度量,其定义如下:

$$IDF_{w_i} = \log \frac{M}{\sum_j |\, j: w_i \in d_j \,|} \tag{2}$$

其中,M 表示文档总数,即所有类别下的文档总数,$\sum_j |\, j: w_i \in d_j \,|$ 表示包含词语 w_i 的文档数目总和。由式(1)、式(2)可得特征词权重公式:

$$TFIDF_{w_{ij}} = TF_{w_{ij}} \times IDF_{w_{ij}} \tag{3}$$

从 TFIDF 的经典公式中,我们可以发现,TF 值越大,即特征词在特征文本中出现的次数越多,权重越大,区分该文本的能力越强。特征项出现在特征文本中的越多,即 IDF 的值越小,表明该特征项区分文本的的能力越弱。IDF 权值部分没有考虑类内特征项的分布规律,这就可能导致在类内分布不均匀的特征项被赋予较高权重。

(2) 考虑类内和类间信息来改进 TFIDF 算法

针对传统算法的不足,很多研究者都对经典的 IFIDF 算法进行了改进。如将信息增益与信息熵融入 TFIDF 算法[11],使用信息熵来调整文本集合内的词条分布,此种方法虽然在一定程度上克服了传统的 IFIDF 算法的不足,但是算法复杂度较高。有学者将 TFIDF 公式改为 $IDF = \log\left(\left(\frac{m}{m+k}\right) \times N\right)$[12],其中 m 为类 C 内包含特征项 w_i 的特征文本的数量,k 为除类 C 外包含该特征项的特征文本的数量。改进的 IDF 的公式在一定程度上克服了传统算法的不足。但是随分类数目和分类文本的数量增加,可能会导致 k 值远大于 m 值,m 的作用微乎其微,可能还会降低 TF 的作用,这样,IDF 就无法表现特征项 w_i 应有的权重。

传统的 TFIDF 特征加权算法单纯地把整个文本集作为一个整体来考虑,其 IDF 部分并不能完全体现特征项的类间分布信息。而往往类间的信息又十分重要,因为类间信息更能代表这个类的特征,更具有代表性。如果使用类间频率 CIF(Classes Internal Frequency)来描述包含特征项的类别的数目,那么在大多数类中出现的特征项对于类别之间的区分能力比较弱,其区分度不大,应该给于其较小的权重。而只在少数类中出现的特征项对于类别之间的区分能力较强,应该赋予较高的权重。为此我们引入逆类间频率 ICIF(Inverse Classes Internal Frequence)来表征特征项对于类别之前的区分能力。它的表达式如下所示:

$$ICIF = \log\left(\frac{C_m}{C_{m_i}} + 1\right) \tag{4}$$

其中,C_m 所有文本的类别总数,C_{m_i} 表示包含第 i 个特征项的类别数目。当特征项只出现在一个类别中的时候,即 $C_{m_i} = 1$,它的 ICIF 值达到最大的 $\lg(C_m + 1)$。根据这个公式的特点,特征项出现的类别数越多,ICIF 的值越小,即该特征项不具有较大的区分能力。这里需要注意的是,ICIF 和 IDF 并不冲突,因为 ICIF 体现的是特征词对类别区分的贡献程度,而 IDF 体现的则是特征项的文本表现能力的熵值这一信息。

但是,此时对于该特征词在类内的分布信息仍然是模糊的,虽然某个特征项的 ICIF 值较大,但是可能该特征项只集中于类内的某篇或者某几篇文档中,并不能代表整个类的特征。TF-IDF 将文档频率 TF 作为个体来考虑,而单个文本中的特征项信息并不能完整体现该类别的全部信息。特征项在某类的内部是否呈高值分布、分布是否均匀都关系到该特征项的权重计算是否合理。为此我们引入类内频率 ICF(Inside Class Frequence),同时为了考虑特征项在整个类内的出现频率,我们也为特征项引入平均类内文档频率 ADFIC(Average Document Frequency Inside Class)的概念,这样即考虑到了特征项在类内部的出现频率,也考虑了特征项在类内的分布特征,公式如下:

$$ADFIC - ICF = \frac{\sum_{C_j} tf_{w_{ij}}}{K_{c_j}} \times \log\left(\frac{nw_i}{nc_j} + 1\right) \tag{5}$$

其中 K_{cj} 表示该类 C_j 中的文档数目,$\sum_{C_j} tf_{w_{ij}}$ 表示特征项 w_i 在类别 C_j 下的文档中总共出现的次数,取平均值是为了计算平均文档频率,防止特征项在某个类别中的某几篇文档中比较集中。而 n_{w_i} 表示特征词 w_i 出现在类别 C_j 下的文档的个数,n_{c_j} 表示类别 C_j 中的总的文档数目,如果 ICF 取值较大,则表明该特征项在类内分布比较均衡。综合起来的式(5)能够很好的反映特征项在类内的分布信息。

使用 IDF 作为平衡因子可以保证分布在较多文本中的特征项的权重较低,即文本表现能力较强,使用 ICIF 可以确保出现在少数类中的特征项获得高权重,而 ADFIC-ICF 能较好地反应特征项在类别内部的分布规律。下面给出修改后的权重计算

公式:

$$W_{w_i} = \log\frac{\sum d_j}{|\{j:w_i \in d_j\}|} \times \log\left(\frac{C_m}{C_{m_i}}+1\right) \times \frac{\sum_{C_j} tf_{w_{ij}}}{K_{c_j}} \times \log\left(\frac{n_{w_i}}{n_{c_j}}+1\right) \quad (6)$$

式(6)中的 $\sum d_j$ 和式(2)中的 M 是一个含义,都代表整个文档集。虽然式(6)较传统的权重计算公式复杂,但得益于 Hadoop 和 Spark 下的高效并发处理模型,式(6)可以得到较高的分类效率和分类准确率。

(3) 改进的特征加权算法在 Spark 中的计算流程

经典的 TFIDF 权值计算在海量文本分类的计算上显得捉襟见肘,往往需要耗费几十个小时,甚至数天时间。这对于实时性较高的场合,显然是无法想象的灾难。为此引入基于内存的分布式计算模型 Spark。式(6)有一个非常显著的特点,三部分权重分值的计算都是独立的,在 Spark 中可以分为三个 Stage,如图 3 所示。

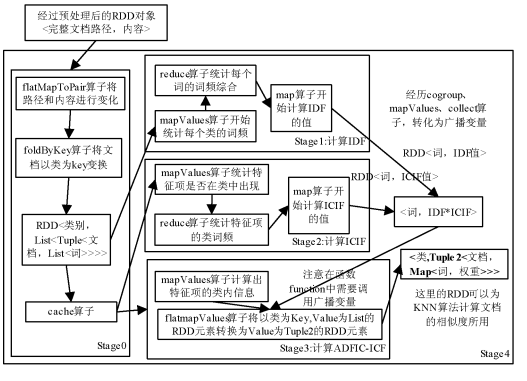

图 3　加权算法在 Spark 中的计算结构图

2.3　文本分类

文本的向量空间模型建立好之后,就需要使用文本分类算法进行分类操作。如前所述,分类算法有很多,K 邻近算法和支持向量机(SVM)等分类算法被认为是分类效果比较好的分类算法[13]。我们这里使用实现比较简单的 KNN 分类算法,在该算法的基础之上设计出基于 Spark 的 KNN 并行化算法。

(1) KNN 分类算法

KNN 算法[14]是一种比较成熟的分类算法,它通过比较测试文档与训练样本集的相似度,找到距离测试文档最近的 K 个文本。当我们计算出每个特征词的权重之后,如果我们把每个文档看成一个多维向量空间,那么该向量由包含所有特征词的权重值组成,该向量的每一维对应一个特征词。文档之间的相似度采用余弦相似度[15]来计算,其计算公式如下所示:

$$sim_{ij} = \cos(d_i,d_j) = \sum_{k=1}^{n} \frac{w_{ik}}{\sqrt{\sum_{k=1}^{n} w_{ik}^2}} \times \frac{w_{jk}}{\sqrt{\sum_{k=1}^{n} w_{jk}^2}} \quad (7)$$

其中,sim_{ij} 表示待测样本 i 和训练样本 j 之间的相似度或者说距离。w_{ik} 表示待测文档 d_i 中的第 k 个特征项 w_k 的权重。而 w_{jk} 表示测试文档 d_i 中的第 k 个特征项在训练文档 d_j 中的权重值。通过循环迭代,计算出待测文档 d_i 和所有训练集中的文档的余弦相似度,并找到与待测文档相似度最高的 K 个训练文档,并计算每个类的权重。测试文档的类别属于分数最高的哪个类别。计算过程如下式所示:

$$score(d_i,c_m) = \sum_{d_j \in c_m} sim(d_i,d_j) - b_m \quad (8)$$

其中 b_m 表示阈值,只考虑分值超过阈值的类别。加入条件 K,使用更通俗的公式来表示:

$$p(d_x,c_j) = \sum_{i=1}^{K} sim(a_i,d_x)p_{a_i,c}(a_i,c_j) \quad (9)$$

其中,$sim(a_i,d_x)$ 表示待测文本 d_x 的 K 个最邻近样本 a_i 和 d_x 之间的相似度,$p_{a_i,c}(a_i,c_j)$ 表示样本 a_i 和类别 c_j 之间的隶属关系,其定义如下所示:

$$p_{a_i,c}(a_i,c_j) = \begin{cases} 1 & a_i \in c_j \\ 0 & \text{其他} \end{cases} \quad (10)$$

计算出每个类相对于待测文本的权重之后,将待测文本归于权重最大的类中,分类也就完成。

(2) KNN 分类算法在 Spark 下的实现

从式(7)可以看出,要得到训练文档 d_i 和测试文档 K 个相似度最高的值,需要使用待测文本 d_x 和整个训练文档集进行相似度的迭代计算。由于整个计算过程是独立的,所以完全可以使用分布式计算模型 Spark 来实现整个计算过程的并行化,如图 4 所示。

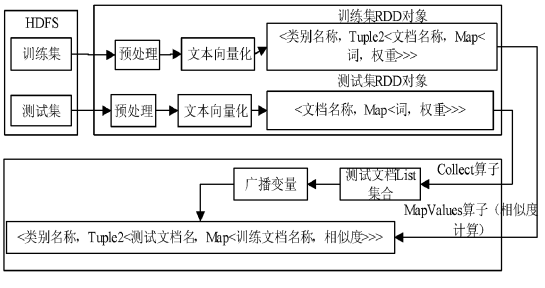

图 4　KNN 分类算法在 Spark 中的计算结构图

(3) KNN 分类算法结果评价

本文采用十折交叉验证来划分训练集与测试集。对于每种加权算法,根据评价指标分别进行 10 次十折交叉验证,并取均值。对于二元分类通常采用的性能评估指标有召回率(Recall,简计为 R)、正确率(Precision,简记为 P)、F1 值以及精确度(Accuracy,简称 A)。一般使用列联表来描述二元分类问题,如表 1 所示。

表 1　二元分类列联表

	实际属于该类文档数	实际不属于该类文档数
属于该类文档数	A	B
不属于该类文档数	C	D

此时,查全率(R)和查准率(P)分类定义为:

$$R = \frac{A}{A+C} \quad P = \frac{A}{A+B}$$

结合召回率和准确率得到 F1 值:

$$F_1(r,p) = \frac{2RP}{R+P} \quad (11)$$

由于正确率、召回率和 F1 值都是针对某一类别进行评价,因此,为了评价分类算法在整个文本集上的性能,通常对单个分类方法的分类指标进行宏平均。计算方法如下所示:

$$MacroP = \frac{\sum_{i=1}^{|C|} P_i}{|C|} \quad MacroR = \frac{\sum_{i=1}^{C} R_i}{|C|} \quad (12)$$

其中的 $MacroR$ 表示宏平均召回率 ,$MacroP$ 表示宏平均准确率。

宏平均是每一个类的性能指标的算术平均值,但是容易受到小类的影响。由于本文研究的是海量 Web 文本的分类,可能每个类别中的文档数目有差别,所以采用精确度作为评估指标,如式(13)所示:

$$A = \frac{A + D}{A + B + C + D} \tag{13}$$

其中左侧的 A 表示精确度,等式的分母其实就是测试文档的总和。

3 实验结果分析和评估

本文使用搜狗实验室提供的新闻语料集作为中文语料实验数据集,其中包含财经、互联网、健康、军事、教育、旅游、体育、文化、环境等 10 大类共 80 000 多篇新闻文本,共计约 108 MB。本文采用的硬件环境是:4 台安装有 Linux 系统的 PC 机,内存大小是 2 GB,四台主机通过路由器相连。本文使用的 Hadoop 版本是 2.6.0,使用的 spark 版本是 1.4.0。4 台主机,其中一台作为 Master 节点,另外四台作为 Slave 节点。Master 运行 NameNode 节点和 Master 进程,Slave 运行 DataNode 和 Worker 进程。对于 Spark 计算模型来说 Master 作为整个集群中的控制器,负责整个集群的正常运行。Worker 相当于计算节点,接收主节点的命令与进行状态汇报。

3.1 分布式节点数和分类时间的关系

本文为了体现计算节点数对分类过程的影响,实验将计算节点数从 1 个逐步增加到 4 个节点,实验结果如表 2 所示。

表 2 Spark 下不同计算节点的训练和测试时间

节点个数(台)	训练时间(s)	测试时间(s)
1	3150	160
2	1826	105
3	1243	72
4	766	46

由表 2 可见:当计算节点只有一个的时候,由于资源(如 CPU、内存)等限制,Spark 的效率还不及本地模式下的计算效率,那是由于 Spark 本身的资源调度需要占用一部分资源和时间,但是随着节点数的增加,实验所需要的训练时间和测试时间均呈明显下降趋势。Web 文本分类系统能快速地完成分类任务。由于实验资源实在有限,计算节点 Worker 的内存只有 2 GB大小,可以预见的是,当集群的内存总量提升时,该文本分类系统的性能还将进一步提升。

3.2 经典 TFIDF 加权算法与改进的加权算法在 Spark 平台下的分类结果

我们在 Spark 计算模型中统计出每次实验的结果,绘制 10 次交叉实验所得的分类正确率曲线图如图 5 所示。

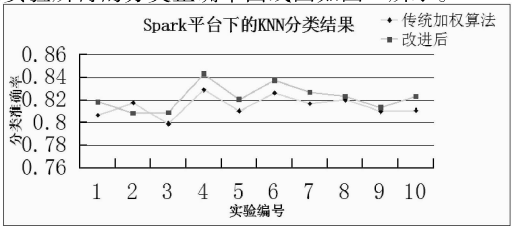

图 5 Spark 平台下 KNN 分类算法的结果

实验证明,通过改进 IDF 算法,充分考虑特征项在类内和类间的的分布规律后,有效地改善了 KNN 分类算法的分类效果。

4 结 语

针对传统的分类方法无法解决大规模分类和实时性分类的不足之处,提出基于 Hadoop 和 Spark 的高效 Web 文本分类系统。该系统充分利用 Spark 计算模型在迭代计算上的优势,对经典的 TFIDF 加权算法进行了优化,充分考虑特征项在整个文本集、语料类的内部以及语料的类别之间的信息,使得整个分类过程高效且无中间结果输出,大大提高了分类的效率。

然而,本文的重点不是研究 Spark,而是为海量信息以及实时性要求很高的文本分类系统提供一种解决方案。本文中的预处理、文本向量化以及 KNN 分类算法在 Spark 上的实现流程还有不完美之处,还有很多值得优化的地方。

参 考 文 献

[1] Rong Lu L I, Yun Fa H U. A Density-Based Method for Reducing the Amount of Training Data in kNN Text Classification[J]. Journal of Computer Research & Development, 2004, 41(4): 539-545.

[2] 程克非,张聪. 基于特征加权的朴素贝叶斯分类器[J]. 计算机仿真,2006,23(10):92-94.

[3] 李荣陆,王建会,陈晓云,等. 使用最大熵模型进行中文文本分类[J]. 计算机研究与发展,2005,42(1):94-101.

[4] Yuan R, Li Z, Guan X, et al. An SVM-based machine learning method for accurate internet traffic classification[J]. Information Systems Frontiers, 2010, 12(2): 149-156.

[5] 丁振国,黎靖,张卓. 一种改进的基于神经网络的文本分类算法[J]. 计算机应用研究,2008,25(6):1640-1641.

[6] 王煜,王正欧. 基于模糊决策树的文本分类规则抽取[J]. 计算机应用,2005,25(7):1634-1637.

[7] 刘文军,郑国义,张小琼. 基于粗糙集与统计学习理论的样本分类算法[J]. 模糊系统与数学,2015,29(1):184-189.

[8] Gao Yanjie. Data Processing with Spark[M]. Beijing: China Machine Press, 2015.

[9] Salton G, Wong A, Yang C S. A vector space model for automatic indexing[J]. Communications of the ACM, 1975, 18(11): 613-620.

[10] Salton G, Buckley C. Term-Weighting Approaches in Automatic Text Retrieval[J]. Information Processing & Management, 1988, 24(88): 513-523.

[11] 李学明,李海瑞,薛亮,等. 基于信息增益与信息熵的 TFIDF 算法[J]. 计算机工程,2012,38(8):37-40.

[12] 张玉芳,彭时名,吕佳. 基于文本分类 TFIDF 方法的改进与应用[J]. 计算机工程,2006,32(19):76-78.

[13] Li Rong, Ye Shiwei, Shi Zhongzhi. SVM-KNN Classifier—A New Method of Improving the Accuracy of SVM Classifier[J]. Acta Electronica sinica, 2002, 30(5): 745-748.

[14] Lihua Sun, Jidong Zhang, Jingmei Li. An improved KNN method and its application to Text classification[J]. Applied Science and Technology, 2003, 29(2): 25-27.

[15] 彭铠,汪伟,杨煜普. 基于余弦距离度量学习的伪邻近文本分类算法[J]. 计算机工程与设计,2013,34(6):2201-2203.