

基于非均衡局部敏感哈希的并行文本分类研究

赵 彤, 刘 斌, 李 涛

(南京工业大学 计算机科学与技术学院, 江苏 南京 210000)

摘 要: 针对 KNN 分类算法在面对海量文本处理时效率低下的问题, 提出了一种基于超平面的非均衡局部敏感哈希分类算法, 该分类算法相比于传统的局部敏感哈希算法在提高分类的准确性和实时性上有显著的效果。同时, 为了进一步降低分类算法的执行时间, 提高分类效率, 将该分类算法与 Spark 并行计算模型结合, 在大数据处理平台 Hadoop 上实现了一种高效的并行文本分类系统。实验结果表明, 所设计的文本分类系统在具有较高分类速度的同时保持了较高的分类准确性。

关键词: KNN; 非均衡局部敏感哈希; Hadoop; Spark

中图分类号: TP31

文献标识码: A

文章编号: 1000—7180(2017)12—0067—07

Research on Parallel Text Classification System Based on Non-Balanced LSH

ZHAO Tong, LIU Bin, LI Tao

(School of Computer Science and Technology, Nanjing Tech University, Nanjing 210000, China)

Abstract: In order to solve the problem of low efficiency of the K-Nearest Neighbors(KNN) classification algorithm in face of massive text, a non-balanced local sensitive hash classification algorithm based on hyper-plane is proposed, which has a more significant effect than the traditional local sensitive hash algorithm on improving the accuracy and real-time performance. At the same time, in order to further reduce the execution time of the classification algorithm and improve the classification efficiency, an efficient parallel text classification system based on Hadoop is designed which combines the classification algorithm and the Spark parallel computing model. The experimental results show that such text classification system has a high classification speed and a high classification accuracy.

Key words: KNN; non-balanced local sensitive hash(NBLSH); hadoop; spark

1 引言

大量研究表明, KNN 分类算法和 SVM(Support Vector Machine)分类算法在中文文本分类上有着优异的性能^[1-2]。KNN 分类算法无严格的训练过程, 实时性高, 但由于待测文本需要计算与所有训练文本集的相似度, 当训练集规模较大且特征文本维度较高时, 算法的时间和空间开销较大。对 KNN 分类算法改进很多, 如文献[3]使用粗糙集进行了改进, 文献[4]提出一种大数据环境下的快速 KNN 分类算法。SVM 分类算法主要适用于小规模语料集, 无法解决大规模分类的问题。当前面对的数据往往

是高维数据, 如何将高维数据转换成低维数据来进行处理是处理高维数据的关键。常用的数据降维技术包括 Z 曲线^[5]、LSH(Local Sensitive Hash)^[6]、iDistance^[7]等等。局部敏感哈希(LSH)算法有效降低了 KNN 分类算法的复杂度, 将待测文本与整体数据集的相似度计算转化为与局部数据集的相似度计算, 效率很高, 但分类准确性有所降低^[8]。很多学者对文本分类算法和局部哈希算法进行了并行化设计, 如文献[9]将基于 LSH 的 K 最邻近算法在 MapReduce 模型下进行了并行化设计, 取得了不错的效果。文献[10]使用 MapReduce 模型和 KNN 算法设计了一种推荐算法。Spark 是一种使用内存计算

引擎的大数据并行处理框架,可以运行在 Hadoop 生态系统之上,兼容 HDFS、Hbase、Hive 等分布式系统,基于内存的特点使得它可以显著提高大规模数据处理的实时性.与 MapReduce 相比,Spark 有着明显的先天优势^[11].

本文在借鉴 E2LSH 哈希算法^[12]和 S2LSH^[13]思想基础上,提出一种新的非均衡局部敏感哈希算法 Non-Balanced LSH(NBLSH),有效改善了使用局部敏感哈希时 KNN 分类算法分类准确性降低的问题.同时,为了进一步提高 NBLSH-KNN 的分类速度,基于 Spark 并行计算模型设计出了一种快速并行分类系统,提高了大规模数据场景下的分类速度.

2 局部敏感哈希算法原理

2.1 局部敏感哈希算法

局部敏感哈希算法的核心思想是,如果两个样本点在原始样本空间中是相邻(相似)的,通过某种哈希映射后,这两个样本点有很大的可能性在新的样本空间也是相邻的.原始数据集中的数据通过哈希函数(Hash Functions)进行映射之后,相邻的样本点有很大的可能性被哈希函数映射到同一个桶(Bucket)中,即通过某种哈希函数进行映射后,元素在哈希表(Hash Table)中的存储位置发生了冲突.哈希函数应该满足如下的条件:

(1) 如果 $d(x, y) \leq d_1$, 则 $h(x) = h(y)$ 的概率至少为 p_1 ;

(2) 如果 $d(x, y) \geq d_2$, 则 $h(x) = h(y)$ 的概率至多为 p_2 ;

式中, $d(x, y)$ 表示样本元素 x 和 y 之间的距离即相似度, $h(x)$ 和 $h(y)$ 分别表示对样本元素 x 和 y 的哈希变换.通常,将满足上述两个条件的哈希函数称之为 (d_1, d_2, p_1, p_2) -sensitive. 而通过一个或多个 (d_1, d_2, p_1, p_2) -sensitive 的哈希函数对原始数据集进行哈希,生成一个或多个哈希表的过程称为局部敏感哈希.

2.2 随机超平面

LSH 条件中的 $d(x, y)$ 表示样本元素 x 和 y 的一个距离度量,但并非所有的距离度量方法都能找到满足 Locality-Sensitive 的哈希函数.基于余弦距离^[14]且满足局部敏感条件的哈希函数通常使用随机超平面来产生,假设样本集合使用 S 来表示,样本集合中的每一个元素都是一个 n 维向量,使用 $\mathbf{v}$ 来表示:

$$\mathbf{v} = \{v_1, v_2, \dots, v_n\} \quad (1)$$

集合 S 中任意两个元素 $\mathbf{v}$ 和 $\mathbf{u}$ 之间的相似度(距离)定义为

$$\text{sim}(\mathbf{v}, \mathbf{u}) = \frac{\mathbf{v} * \mathbf{u}}{|\mathbf{v}| |\mathbf{u}|} \quad (2)$$

设 $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ 是 n 维向量空间中的一个随机向量,如果以向量 $\mathbf{x}$ 作为法向量产生一个随机平面 P_0 ,那么 P_0 将原始 n 维空间分成了不同的两个部分,如图 1 所示.同理可以产生 P_1 和 P_2 .

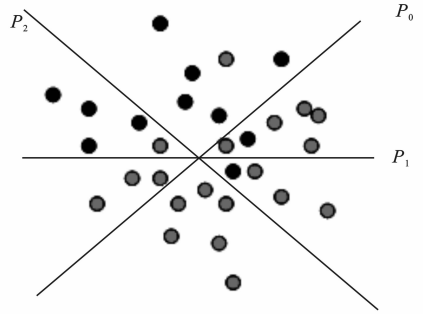


图 1 超平面和样本空间示意图

如果规定法向量指向的一侧空间中的点被哈希成 1,而相反的一侧被哈希成 0.给定一个未知向量 $\mathbf{v}$,可以通过计算 $\mathbf{v}$ 与 $\mathbf{x}$ 的夹角来判定该未知向量 $\mathbf{v}$ 属于哪半部分空间,具体公式定义如下:

$$h(\mathbf{v}) = \begin{cases} 1, & \mathbf{v} * \mathbf{x} > 0 \\ 0, & \mathbf{v} * \mathbf{x} \leq 0 \end{cases} \quad (3)$$

从式(3)分析得到,假设向量 $\mathbf{v}$ 和 $\mathbf{u}$ 的夹角是 θ ,由于法向量 $\mathbf{x}$ 的随机性,两个不在超平面 P_0 两侧的概率为 $p(\theta) = 1 - \frac{\theta}{\pi}$.此时假设向量 $\mathbf{v}$ 和 $\mathbf{u}$ 的相似度是 S ,根据 $\theta = \arccos(S)$ 可以得出概率公式(4):

$$p(s) = 1 - \frac{\arccos(s)}{\pi} \quad (4)$$

通过式(3)还无法产生所需的哈希值,因其哈希值只有两个,通常需要产生多个独立的哈希值进行级联,如式(5)所示.

$$H(\mathbf{v}) = (h_1(\mathbf{v}), h_2(\mathbf{v}), \dots, h_k(\mathbf{v})) \quad (5)$$

从式(5)可以看出,级联之后两个向量哈希值相等的概率如式(6)所示,此时哈希值的个数变为了 2^k .

$$p(s) = \left(1 - \frac{\arccos(s)}{\pi}\right)^k \quad (6)$$

3 非均衡局部敏感哈希 NBLSH

3.1 NBLSH 的原理

通过使用 LSH 函数簇中的哈希函数对原始样

本集合进行映射,保证了原始数据空间中距离相近点的冲突概率大于距离较远点的冲突概率.通常,为了更加有效地寻找到与待测样本点最近的训练样本点,提高邻近点寻找的准确率,需要同时创建多个哈希表,如图2所示.

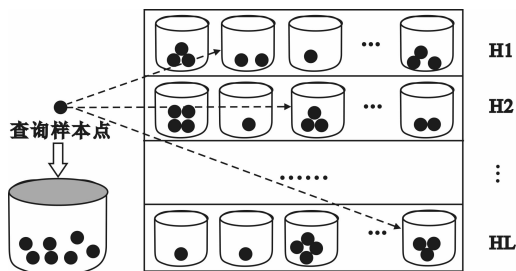


图2 多局部哈希映射示意图

分析图2可知,某个样本点 q_i 会同时落到 L 个 Bucket 中,并且由于产生每个哈希表的哈希函数是随机的,可能导致 q_i 所在的哈希桶中样本点数量不同.如果哈希桶 B_i 使用区域 A_i (Area _{i}) 来表示,那么待测样本点与所有包含该样本点的区域的示意图如图3所示.

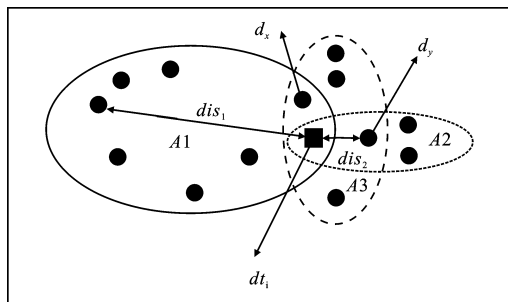


图3 多哈希表样本点分布示意图

图3中的 dis_1 表示待测样本点 dt_i 与区域 A_1 中某样本点的距离,而 dis_2 表示待测样本点 dt_i 与区域 A_2 中某样本点的距离.从中可以看出,区域 A_1 、 A_2 、 A_3 都包括了待测样本点 dt_i ,在图1中表现为待测样本点落到了哈希表 H_i 的桶 B_i 中.分析图3可知,区域 A_2 和 A_3 相比于 A_1 ,面积较小,所以待测样本点 dt_i 与 A_2 和 A_3 中样本点的距离普遍低于 A_1 ,即 A_2 和 A_3 中的样本点更应该作为待测样本点 dt_i 的候选 K 最邻近点.换句话说,如果 dt_i 所在的哈希桶 B_i 的“容量”越小,那么该桶中的样本点作为 dt_i 的候选 K 最近邻近点的概率就越大.因此,为了提高 K 最邻近点寻找的准确率,更应该使用“容量”较小的桶中的样本点.反之,如果桶的容量越大,可能包含较多与 dt_i 的距离较远的样本点.

如果桶容量(区域大小)越小,待测样本点 dt_i

越有可能成为 K 最邻近候选样本点.在样本空间划分上表现为随机超平面的数目越多,划分的空间区域越小,在哈希表中表现为桶的容量越小.但是,空间区域越小,会导致与 dt_i 邻近的样本点个数较少,如果训练样本点经过哈希函数映射到某个桶中后,该桶中的样本密度和总数均较小时,需要创建更多的哈希表才能提供一定数量的 K 最邻近候选点,这会导致创建哈希表的存储空间不断扩大,不适合进行规模数据的索引,并且通过牺牲存储空间换取的准确率提升叫较为缓慢.为此,本文提出非均衡局部哈希表(NBLSH)的概念,每张哈希表的大小是不一样的,表现为产生每张哈希表所用的超平面的个数是不相同的,即每张哈希表对应的桶的容量大小不同.但必须满足,桶容量较小的哈希表的数量占多数,而容量较大的占少数.

假设某划分样本空间需要最少的随机超平面的个数为 N_1 ,最大数目为 N_2 ,且一开始产生的随机超平面的个数为 N . N 、 N_1 和 N_2 须满足下列条件:

$$\begin{cases} N_2 > N_1 \\ N > x \cdot N_2 \end{cases} \quad (N_1, N_2, N \text{ 均大于 } 0, x \geq 5) \quad (7)$$

N 的值应该取 N_2 的 3~5 倍以上,这样可以以更高的随机性获取随机超平面.由于随机超平面是由法向量决定的,故预先产生的 N 个随机超平面的法向量形成了一个向量池.设 n_i 是介于 N_1 和 N_2 之间的一个随机数,表示每次从向量池取出 n_i 个随机向量,产生哈希表的哈希函数 g_i 由这 n_i 个随机向量决定,故产生不同哈希表的映射函数 g_i 将会决定最终哈希桶的容量的大小.

由于 n_i 的随机性,可能导致 n_i 全部偏大或者全部偏小,这可能导致候选点产生的结果不确定性增加.通过前面的分析可知,应当尽量多地创建哈希桶容量小的哈希表,而尽量少地创建哈希桶容量大的哈希表.该随机数产生特征刚好满足偏态分布的特点,但目前 Spark MLlib 还无法产生偏态分布的随机数,只能产生标准正态分布并通过 RDD 转换为非标准正态分布.文本使用正态分布近似替代偏态分布来产生随机数,如图4所示.

若特征向量空间中特征向量的维度数 $d=10$,最小随机超平面数 $N_1=30$,最大随机超平面数 $N_2=100$,则使用 Spark 的 MLlib 中的 RandomRDDs 的工厂方法产生 5 次随机变量,结果如表1所示.

3.2 最优 K 最邻近候选点的产生

设分类系统使用的哈希表的个数为 L ,对应的哈希表使用 H_i ($0 < i < L$) 来表示,哈希表集合 H

$= \{h_i \mid i = 1, 2, 3, \dots, L\}$. 产生哈希表 H_i 的哈希函数使用 g_i 来表示, 整个哈希函数集合 $G = \{g_i \mid i = 1, 2, 3, \dots, L\}$. 待测样本点 dt_i 经过哈希函数 G 映射后, 落入到了哈希表 H_i 的哈希桶 B_i 中, 为了方便描述, 将哈希桶使用平面来描述, 即使用区域 A_i 表示 B_i . 如果将区域 $A_1 \sim A_L$ 中的所有样本点作为 K 最邻近候选点, 那么从中获取 K 个最邻近点的计算开销仍然很大, 特别是当训练样本集很大的时候, 每个区域中的样本点的个数仍然很大. 根据图 3 的分析, 并不是每个区域中的样本点与待测样本点 dt_i 的距离都很近, 特别当某些样本点处于区域的边界时, 实际与 dt_i 的距离很远, 应该不予考虑. 故不应该选取所有的区域, 而是取那些最大可能包含 K 最邻近候选点的区域.

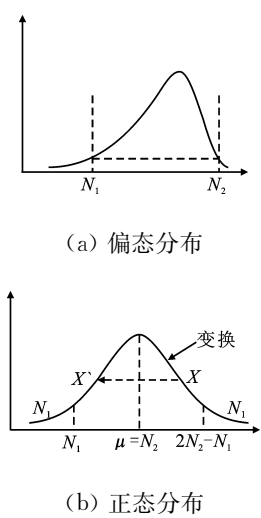


图 4 偏态分布与正态分布

表 1 Spark 产生正态分布随机数

$N_1=30 \quad N_2=100 \quad \sigma=20$				
92	78	30	38	48
30	95	89	72	60
57	64	71	30	95
81	90	52	34	94
30	30	83	39	80
80	75	88	52	86
74	50	58	76	79
76	84	60	69	96
96	95	30	90	48
65	71	80	76	93

如果将每个区域 A_i 看成整体, 根据某些条件计算出这些区域的权重, 权重越大的区域应该首先被考虑, 为此本文引入候选区域 A_i 的权重计算公式, 如下:

$$W(A_i, q) = \log \left[\frac{Hyp(A_i)}{\sum_{i=0} q_i} \right] \cdot \frac{1}{AVG_{dis}(q, q_i)} \quad (q \in A_i, q_i \in A_i) \tag{8}$$

式中, $Hyp(A_i)$ 表示区域 A_i 对应的哈希函数 g_i 使用的超平面的个数, 它的值越大, 表明哈希桶 B_i 的容量越小, 对应的区域 A_i 的面积越小, 权重应该较高. $\sum_{i=0} q_i$ 表示区域 A_i 中的样本点数, 该值越高, 表明区域 A_i 的面积越大, 其权重应该较低. $AVG_{dis}(q, q_i)$ 表示待测点 q 与区域 A_i 中的样本点的平均距离, 如果 $AVG_{dis}(q, q_i)$ 的值越小, 表明区域 A_i 中的样本点与 q 平均越接近, 也就越有可能作为 q 的 K 最邻近的候选点. 假设需要获取的候选区域 A_i 的数目为 M , 那么需要在所有候选区域中取权重值最高的 M 个区域中的样本点作为 K 最邻近候选点. M 的大小取决于 KNN 算法中的 K 值以及文本分类系统的实时性要求.

最后, 将权重最高的 M 个候选区域进行合并, 作为最终的候选区域 A , 并计算区域 A 中所有的样本点与待测样本点 q 的余弦距离, 最后获取前 K 个样本点作为最终的 K 最邻近点.

4 基于 NBLSH-KNN 的算法步骤

(1) 构造非均衡哈希表 NBLSH 算法

输入: 设训练文本集合构成的向量空间是 d 维的, 即特征空间总共有 d 个特征项构成, 训练文本集合 $V = \{v_i \mid i = 1, 2, 3, \dots, n\}$. 需要生成哈希表^[15] 的个数 L , 需要产生随机超平面的个数为 N , 构造哈希函数最小和最大的超平面的个数分别为 N_1 和 N_2 .

输出: 生成的哈希表集合 $T = \{t_i \mid i = 1, 2, 3, \dots, l\}$ 和哈希表中使用的哈希函数集合 $G = \{g_i \mid i = 1, 2, 3, \dots, l\}$. 其中每个哈希函数由 n_i 个超平面产生, 其中 n_i 属于 (N_1, N_2) ;

步骤 1 随机产生 N 个超平面的随机法向量组成向量池, 每个随机法向量的每个维度上的值是通过高斯分布随机产生的. 对第 i 个哈希表从向量池中获取 n_i 个法向量, 它们决定了 n_i 个不同的超平面, 由此产生哈希函数 g_i , 并组成哈希函数集合 G .

步骤 2 对于训练样本集合中的每个样本 $v_i (i = 1, 2, 3, 4 \dots, n)$ 使用哈希函数 g_j 进行计算, $g_j(v_i)$ 的返回值就是该样本 v_i 在第 j 个哈希表 t_j 中的存储位置, 即样本 v_i 落入到的哈希桶的位置.

(2) 使用 NBLSH 哈希表进行分类算法

输入:训练样本集合: $V = \{v_i \mid i = 1, 2, 3, \dots, n\}$, 哈希表集合 $T = \{t_i \mid i = 1, 2, 3, \dots, l\}$, 哈希函数集合 $G = \{g_i \mid i = 1, 2, 3, \dots, l\}$, 待分类样本集合 $Vt = \{v_i \mid i = 1, 2, 3, \dots, m\}$.

输出: 分类结果集合 $R = \{r_i \mid i = 1, 2, 3, \dots, m\}$

步骤 1 待测样本集合中的每个待测样本 v_i 使用哈希函数 g_j 进行运算, g_j 的返回值就是该待测样本 v_i 在第 j 个哈希表 t_j 中的哈希桶的位置.

步骤 2 取出步骤 1 中得到的第 j 个哈希表 t_j 中的哈希桶 B_j 中的所有样本子集, 这些样本子集分别组成了不同的候选区域 A_j , 分别计算这些候选区域 A_j 的权重 W_j , 取出权重最高的 M 个候选区域.

步骤 3 将步骤 2 中得到的 M 个候选区域中的样本集合进行合并组成一个大的区域 A , 别计算待

测 v_i 与 A 中所有样本点的余弦相似度, 得到余弦相似度最高的 k 个样本组成集合 S . 最后按照 KNN 分类原理得到该样本 v_i 属于的类别.

5 NBLSH-KNN 并行化设计

Spark 是 UC Berkeley AMP lab 所开源的集群计算平台, 与 Hadoop 有相似之处, 但它是立足于内存计算的一个新的集群计算框架, 具体见图 5. Hadoop 的数据共享相比 Spark 要慢, 因为前者需要额外的复制、序列化和磁盘 IO 带来的开销, 而后者是基于内存计算且作业是基于构建的 Stage 有向无环图(DAG)执行的. 由于 Spark 的集群架构设计和内核处理模式, 使得其更适合进行大数据挖掘或者机器学习处理等需多次迭代的领域, 因为 Spark 设计中融入了 RDD 这样的抽象概念.

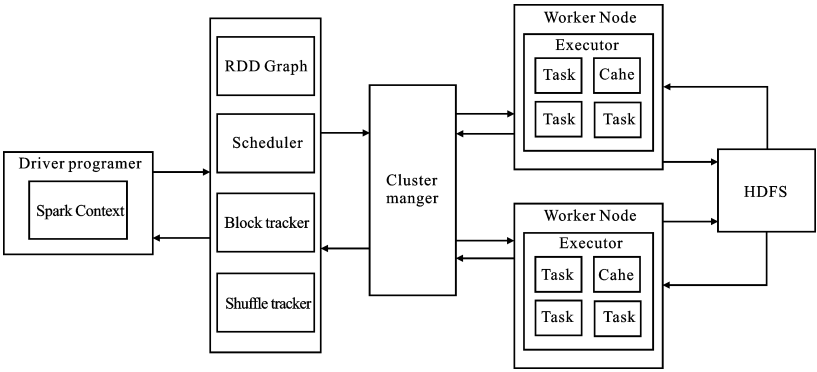


图 5 Spark 分布式集群架构

RDD(Resilient Distributed Dataset)是 Spark 的核心概念, 指一个只读的、可分区的弹性分布式数据集. Spark 默认的缓存策略是将 RDD 以反序列化的方式缓存于内存中, RDD 的 Lineage 会记录每一个 RDD 的父依赖, 也就是 Transformation 操作行为. 当这个 RDD 的部分分区数据丢失时, 由于 Spark 的幂等特性以及函数式语义的设计, 它可以通过 Lineage 获取依赖信息去重新运算并恢复丢失的数据分区. Spark 的这种粗粒度数据并行的计算范式, 带来了性能的提升并保证了数据的鲁棒性.

6 实验结果及分析

实验中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, 其中包含环境、IT(计算机)、交通等 10 大类文本数据, 每个类下包含 8 000 篇左右的文档. 使用的集群硬件环境是: 4 台 PC 机, 内

存大小是 2 GB, 四台主机通过路由器相连, 由于实验条件的限制, 本系统只搭建了 4 个节点的 Spark 集群环境. 整个集群采用 Spark On YARN 的架构, 即 Spark 运行在 Hadoop 平台之上.

为了验证改进的局部敏感哈希算法 NBLSH 在提高 K 最邻近点寻找上的时间优势, 实验将 10 个类别下的文档数分别从 500 提升到 4 000, 即训练样本总数从 5 000 提升到 40 000, 观察不同的算法在 Spark 下的执行时间, 如图 6 所示. 需要注意的是, 文本预处理和文本生成向量的时间不在观察范围之内, 执行时间指的是待查样本点进入哈希表中、寻找 K 个最邻近点以及最终判定待测点所属类别过程所需要的时间.

从图 6 中可以看出, NBLSH 算法在三种算法中所需要的执行时间是最短的. 当训练语料集规模较小时, NBLSH 算法和 KNN 算法以及 E2LSH 算法所需的时间相差不大. 此时, 由于 K 值的限制,

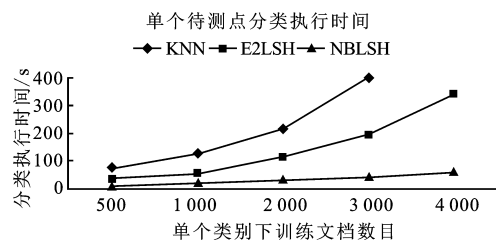


图 6 分类执行时间

E2LSH 和 NBLSH 选择的候选 K 最邻近点的数目相差不大.但随着文本规模的增加,E2LSH 的优势越发明显,表现为折线间的高度差越来越大.对于 E2LSH 而言,它需要计算与所有候选区域中所有的样本点的距离,而 E2LSH 只需要选择权重较高的候选区域,将待测点与这些区域中的样本点计算距离即可.

为了验证 NBLSH 算法相比于传统的 LSH 算法在提高分类准确性的优势,本文从语料集每个类中分别截取 200 篇文档进行分类效果的统计.实验先统计出了全部待测文本的分类准确率,结果如图 7 所示.

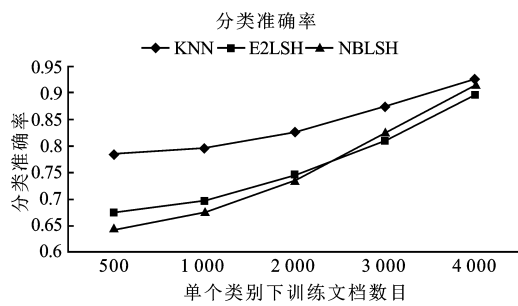


图 7 分类准确率

根据图 7 所示的结果可知,KNN 分类算法中的 K 最邻近寻找在使用局部敏感哈希算法进行优化之后,分类准确率有一定降低.当训练样本集规模较大时,本文提出的 NBLSH 算法可以在提高分类执行时间的同时,最大程度提高分类准确率,最大程度接近 KNN 算法的执行时间.

实验在训练文档取最大值 40 000 时,又针对每个类别分别统计出了三种算法的执行时间以及查准率和查全率,如表 2 和表 3 所示.

表 2 Spark 平台下不同分类算法的时间对比

节点个数/ 台	KNN/ s	LSH-KNN (E2LSH)/s	NBLSH-KNN/ s
1	3 250	1 361	462
2	1 838	785	312
3	1 243	572	172
4	846	348	118

需要注意的是,表中记录的执行时间不含预处理

时间,即文本向量化和哈希表创建所需时间不包含在其中.根据表 3 的统计结果可知,NBLSH-KNN 算法相比于原始 KNN 和 LSH-KNN 分类算法需要更少的分类时间,并且通过计算可知,NBLSH 在 Spark 集群上的加速比高于 KNN 和 LSH-KNN 算法,即当集群数量增加时,性能提升更快.

表 3 各个类的分类结果统计 %

类别	KNN		E2LSH-KNN		NBLSH-KNN	
	查全率 (R)	查准率 (P)	查全率 (R)	查准率 (P)	查全率 (R)	查准率 (P)
交通	88.25	86.12	76.18	81.33	86.18	84.33
体育	85.68	91.58	78.93	86.78	85.93	89.78
医药	82.22	86.64	67.27	77.43	79.27	80.43
政治	72.46	68.46	62.46	54.96	72.46	66.96
经济	77.68	72.46	54.68	49.68	74.68	71.68
艺术	69.32	64.75	62.58	58.80	62.58	58.80
军事	91.45	89.34	84.34	81.24	89.34	89.24
IT	90.10	89.90	76.12	82.90	88.12	87.90
环境	76.00	86.90	62.78	82.26	74.78	84.26
教育	78.57	78.32	76.32	74.92	76.32	74.92

从表 3 中可以看出,三种算法在大类上具有较好的分类效果,如交通、军事、计算机等.而在类别区分度不大的类别上效果不是很理想,如政治、艺术、经济等.进一步分析可知,虽然 LSH-KNN 算法相比于传统 KNN 分类算法有一定时间性能上的优势,但其降低了各个类的查准率和查全率.由于 NBLSH-KNN 分类算法很大程度上改善了 K 最邻近点的寻找效果,相比于 LSH-KNN 算法,分类准确率更加接近于原始 KNN 分类算法.

7 结束语

本文基于 Spark 并行计算模型,实现了一种高效的文本分类系统.该文本分类系统对传统的局部敏感哈希算法进行了改进,更加精确地匹配到待测文本的 K 个最邻近点,便于 KNN 算法进行相似度计算.由于实验条件的限制,本文的数据还达不到 GB 或 TB 级,但该并行文本分类系统对大数据情况下文本分类提供了一种思路和解决方案,具有一定的工程应用价值.

参考文献:

[1] 刘怀亮,张治国,马志辉,等. 基于 SVM 与 KNN 的中
文文本分类比较实证研究[J]. 情报理论与实践,
2008, 31(6): 143-146.
[2] 马建斌,李滢,滕桂法,等. KNN 和 SVM 算法在中文

- 文本自动分类技术上的比较研究[J]. 河北农业大学学报, 2008, 31(3): 120-123.
- [3] 余鹰, 苗夺谦, 刘财辉, 等. 基于变精度粗糙集的 KNN 分类改进算法[J]. 模式识别与人工智能, 2012, 25(4): 617-623.
- [4] 苏毅娟, 邓振云, 程德波, 等. 大数据下的快速 KNN 分类算法[J]. 计算机应用研究, 2016, 33(4): 1003-1006.
- [5] Yao B, Li F, Kumar P. K nearest neighbor queries and knn-joins in large relational databases (almost) for free[C]//Data engineering (ICDE), 2010 IEEE 26th international conference on. IEEE, 2010: 4-15.
- [6] Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality[C]//Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM, 1998: 604-613.
- [7] Jagadish H V, Ooi B C, Tan K L, et al. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search[J]. ACM Transactions on Database Systems (TODS), 2005, 30(2): 364-397.
- [8] Datar M, Immorlica N, Indyk P, et al. Locality-sensitive hashing scheme based on p-stable distributions [C]//Proceedings of the twentieth annual symposium on Computational geometry. ACM, 2004: 253-262.
- [9] Zhu P, Zhan X, Qiu W. Efficient k-Nearest Neighbors Search in High Dimensions Using MapReduce[C]//Big Data and Cloud Computing (BDCLOUD), 2015 IEEE Fifth International Conference on. IEEE, 2015: 23-30.
- [10] 陈佑雄, 向阳, 张骐, 等. 基于 LSH 和 MapReduce 的近邻模型推荐算法[J]. 微电子学与计算机, 2013, 30(12): 47-49.
- [11] Gao Yanjie. Data Processing with Spark[M]. Beijing: China Machine Press, 2015.
- [12] Andoni A, Indyk P. E2lsh: Exact euclidean locality-sensitive hashing [EB/OL]. [2017-01-09]. <http://web.mit.edu/andoni/www/LSH/>. 2004.
- [13] Park Y, Hwang H, Lee S. A Fast k-Nearest Neighbor Search Using Query-Specific Signature Selection [C]//Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. ACM, 2015: 1883-1886.
- [14] 彭凯, 汪伟, 杨煜普. 基于余弦距离度量学习的伪 K 近邻文本分类算法[J]. 计算机工程与设计, 2013, 34(6): 2200-2203.
- [15] 戴上平, 冯鹏, 刘盛英杰, 等. 基于余弦距离的敏感哈希的 KNN 算法在中文文本上的快速分类[J]. 计算机工程与科学, 2015, 37(10): 1971-1976.

作者简介:

赵彤男, (1992-), 硕士研究生. 研究方向为数据挖掘.
E-mail: laokuazi@163.com.

刘斌男, (1966-), 博士, 硕士生导师. 研究方向为模糊技术、人工智能、数据挖掘.

李涛男, (1990-), 硕士研究生. 研究方向为数据挖掘.

(上接第 66 页)

参考文献:

- [1] Luo J, Fan L, Chen Z, et al. A solid state drive architecture with memory card modules[J]. IEEE Transactions on Consumer Electronics, 2016, 62(1): 17-22.
- [2] Bez R, Camerlenghi E, Modelli A, et al. Introduction to flash memory[J]. Proceedings of the IEEE, 2003, 91(4): 489-502.
- [3] Seong Y J, Nam E H, Jin H Y, et al. Hydra: A block-mapped parallel flash memory solid-state disk architecture [J]. IEEE Transactions on Computers, 2010, 59(7): 905-921.
- [4] 雷雨. 基于 PCIE 固态硬盘的嵌入式存储技术[D]. 北京: 中国科学院, 2014.
- [5] Kim H, Choi H, Kim Y et al. MS2: NAND-flash SSD-based multi-layered storage management system [C]//2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). Chengdu, 2016: 100-104.
- [6] Huh J, Kim J S, Jung D, et al. Parameter-aware I/O management for solid state disks (SSDs) [J]. IEEE Transactions on Computers, 2012, 61(5): 636-649.
- [7] Cai Y, Haratsch E F, McCartney M, et al. FPGA-based solid-state drive prototyping platform [C]//IEEE, International Symposium on Field-Programmable Custom Computing Machines. IEEE, 2011: 101-104.
- [8] 李硕, 刘芳, 肖依. 基于 FPGA 的 PCIe SSD 设计与实现[J]. 中国科技论文, 2014, 19(4): 403-406.

作者简介:

宋鹏程男, (1993-), 硕士. 研究方向为数字电路设计.
E-mail: songpc14@mails.tsinghua.edu.cn.

张春男, (1972-), 博士, 副研究员. 研究方向为 RFID 技术与新型触觉传感器.