

基于 MapReduce 和矩阵的频繁项集挖掘算法

周国军, 龚榆桐

(玉林师范学院 数学与信息科学学院, 广西 玉林 537000)

摘 要: 为了能高效地从大数据集中挖掘所有频繁项集, 提出了一种基于 MapReduce 和矩阵的频繁项集挖掘算法. 该算法将事务数据库转化为矩阵, 按照垂直划分方法将矩阵分解成多个规模相同的子矩阵, 根据 MapReduce 模型将子矩阵分配给计算机集群的节点, 各节点并行对子矩阵计算候选项集的支持度. 该算法在执行过程中产生的通信量较少, 实现了节点计算任务的负载均衡. 在 Hadoop 平台上测试了算法的性能, 实验结果表明该算法具有较好的加速比和可扩展性, 适合对大数据集挖掘频繁项集.

关键词: MapReduce; Hadoop 平台; 矩阵; 频繁项集; 关联规则

中图分类号: TP311.1

文献标识码: A

文章编号: 1000-7180(2016)05-0119-05

Frequent Itemsets Mining Algorithm Based on MapReduce and Matrix

ZHOU Guo-jun, GONG Yu-tong

(School of Mathematics and Information Science, Yulin Normal University, Yulin 537000, China)

Abstract: To efficiently find all frequent itemsets from large data sets, a frequent itemsets mining algorithm based on MapReduce and matrix is proposed. The algorithm includes the following points: Transaction database is transformed into matrix, then the matrix is divided into multiple submatrices by vertical partitioning method. According to MapReduce model, submatrices are assigned to the nodes of cluster, and the support of candidate itemsets is parallely computed by the nodes. The amount of communication generated during the execution of the algorithm is small. Load balance among nodes is achieved. The performance of the algorithm is tested on Hadoop platform. Experimental results show that the algorithm has good speedup and scalability, which is suitable for mining frequent itemsets from large data sets.

Key words: MapReduce; Hadoop platform; matrix; frequent itemsets; association rules

1 引言

自从 R. Agrawal 和 R. Srikant 提出经典的 Apriori 算法^[1]以来, 已经有很多文献提出了基于矩阵的频繁项集挖掘算法. 与 Apriori 算法相比, 这些算法将对事务数据库的扫描转化为对矩阵的计算, 缩短了 I/O 时间, 提高了发现频繁项集的效率^[2]. 面对日益增长的海量数据, 串行的基于矩阵的频繁项集挖掘算法面临着两个难以解决的问题: 一是受单机内存的限制, 难以将表示大数据集的矩阵存放

于内存中; 二是受单机计算能力的限制, 从大矩阵中找出所有频繁项集需要较长的计算时间. 针对这两个问题, 本文根据 Hadoop^[3] 平台的 MapReduce^[4] 模型, 提出了一种基于 MapReduce 和矩阵的频繁项集挖掘算法. 该算法将事务数据库转化为矩阵, 按照垂直划分方法将矩阵分解成多个子矩阵, 各节点并行对子矩阵计算候选项集的支持度. 该算法利用了云计算平台海量的内存空间和强大的计算能力, 解决了单机内存不能存放大矩阵的问题, 提高了对大数据挖掘频繁项集的效率.

收稿日期: 2015-06-20; **修回日期:** 2015-08-11

基金项目: 广西高校科学技术研究立项项目(LX2014300)

2 相关知识及定义

2.1 频繁项集的定义

设 $I = \{I_1, I_2, \dots, I_m\}$ 是由 m 个项组成的集合, 事务数据库 D 是由 n 个事务组成的集合. 事务 T 表示为 $T = \langle \text{TID}, \text{Items} \rangle$, TID 是事务的标识符, Items 是事务的项列表并且 $\text{Items} \subseteq I$. 为了便于算法描述, 假定事务或项集中的项按字典次序排序.

定义 1 设 $X (X \subseteq I)$ 是一个项集, 事务数据库中包含 X 的事务数称为 X 的支持度计数, X 的支持度计数与总事务数的百分比称为 X 的支持度. 如果 X 的支持度不小于预定义的最小支持度阈值, 则称 X 是频繁项集.

定义 2 包含 k 个项的项集称为 k -项集. 对于任意一个 k -项集 c_k , 其向量表示为 $v = (a_1, a_2, \dots, a_n)$, 其中, n 是事务数, 如果事务数据库中的第 j 个事务包含 c_k , 则对应的第 j 个元素 $a_j = 1$, 否则 $a_j = 0$. 项集 c_k 对应向量的所有元素之和即为 c_k 的支持度计数^[5], 表示为

$$\text{sup_count}(c_k) = \sum_{j=1}^n a_j, a_j \in v.$$

定义 3 对于任意两个频繁 k -项集 l_1 和 l_2 , 如果 $l_1[1 \dots k-1] = l_2[1 \dots k-1]$ 并且 $l_1[k] < l_2[k]$, 则 l_1 和 l_2 能够连接生成候选 $(k+1)$ -项集 c_{k+1} , 即 $c_{k+1} = l_1 \circ l_2 = \{l_1[1], l_1[2], \dots, l_1[k], l_2[k]\}$. 如果 l_1, l_2 对应的向量分别为 v'_1, v'_2 , 则 c_{k+1} 的向量 $v = v'_1 \wedge v'_2$, 其中“ $\wedge$ ”表示逻辑与运算^[5].

定义 4 所有候选 k -项集的向量构成了一个 r 行 n 列的矩阵, 表示为 A_k , 其中, r 是候选 k -项集的数量, n 是事务数.

2.2 Hadoop 平台的 MapReduce 模型

MapReduce 是 Google 提出的一种并行编程模型, 核心是 Map 函数和 Reduce 函数, 它们通过操作键/值对来处理大数据集^[6]. Hadoop 是 Apache 开发的云计算平台, 该平台实现了 MapReduce 模型. Hadoop 的 MapReduce 框架采用了主/从式结构, 主节点称为 JobTracker, 从节点称为 TaskTracker. 在处理大数据的过程中, MapReduce 框架将大数据集分解成多个数据分片, 主节点将计算任务和数据分片分配给多个从节点, 从节点通过并行执行 Map 和 Reduce 任务完成对数据分片的处理.

在 Hadoop 中, 主函数、Map 函数和 Reduce 函数组成了 MapReduce 应用程序的基本结构. 作业 (Job) 的配置和启动、程序的流程控制由主函数实

现, 具体的计算任务则由 Map、Reduce 函数实现. Hadoop 平台提供了 Map、Reduce 函数的模板, 这两个函数的功能由用户编写的 Java 类实现. 在 Map 函数中可以定义 setup、map 和 cleanup 等方法, 相应地, 在 Reduce 函数中可以定义 setup、reduce 和 cleanup 等方法. 对于每一个 Map 或 Reduce 任务, setup 方法在任务启动后被调用一次, cleanup 方法在任务结束前被调用一次. map 方法与 reduce 方法的数据流^[7]表示如下:

map: $(K_1, V_1) \rightarrow \text{list}(K_2, V_2)$

reduce: $(K_2, \text{list}(V_2)) \rightarrow \text{list}(K_3, V_3)$

在该数据流中, (K_1, V_1) 是数据分片的一条记录由 MapReduce 框架解析得到的键/值对, 在默认的情况下, K_1 是一行记录在数据分片中的字节偏移量, V_1 是一行记录的具体内容. $\text{list}(K_2, V_2)$ 是 map 方法输出的键/值对列表, $\text{list}(K_3, V_3)$ 是 reduce 方法向 Hadoop 分布式文件系统 HDFS 输出的结果.

3 基于 MapReduce 和矩阵的频繁项集挖掘算法

3.1 算法的基本思想

(1) 使用矩阵表示所有候选 k -项集的向量, 在迭代计算频繁 $(k+1)$ -项集的过程中不断缩减矩阵的规模, 减少算法的 I/O 时间. 使用逻辑与运算计算候选 $(k+1)$ -项集的向量, 进而计算候选项集的支持度. 由于逻辑与运算速度大于扫描事务数据库的模式匹配速度, 减少了算法的执行时间.

(2) 采用水平划分方法分解矩阵会造成频繁项集的缺失^[8], 将计算任务均衡分配给各节点是提高大数据处理效率的重要因素^[9]. 基于这两个要点, 算法所使用的矩阵分解方法如下: 根据从节点的个数 y , 将矩阵垂直划分为 s 个子矩阵, 其中, s 等于 y 或者 s 等于 y 的整数倍. 前 $s-1$ 个子矩阵的列数均为 $\text{ceil}(n/s)$, 最后一个子矩阵的列数为 $n - (s-1) \times \text{ceil}(n/s)$, 其中 ceil 表示向上取整.

(3) 各节点对子矩阵并行计算候选项集的支持度, 为了减少节点间的通信量, Map 任务计算候选 $(k+1)$ -项集 c_{k+1} 的局部支持度, 并将 c_{k+1} 的向量保存到 HDFS 中.

3.2 算法的主要步骤和描述

算法所使用的符号说明如下: 最小支持度阈值用 minsup 表示, 事务数用 n 表示, 事务数据库的第 j 个事务的 TID 用 j 表示, 即 $\text{TID} = j$. 子矩阵的个数

用 s 表示, $\mathbf{A}_k$ 的 s 个子矩阵分别表示为 $\mathbf{A}_{k1}, \mathbf{A}_{k2}, \dots, \mathbf{A}_{ks}$, 保存子矩阵 $\mathbf{A}_{ki} (1 \leq i \leq s)$ 的文件名用 $f_i (1 \leq i \leq s)$ 表示. 算法的主要步骤如下所示.

(1) 主函数将事务数据库的数据分片分配给执行 Map 任务的节点.

(2) 计算频繁 1- 项集. 具体过程是: Map 函数读取数据分片, 将每个事务 T 包含的候选 1- 项集 c_1 分别以 $\langle c_1, \text{TID} \rangle$ 键 / 值对输出到 Reduce 函数. Reduce 函数对 $\langle c_1, \text{list}(\text{TID}) \rangle$ 计算 c_1 的支持度计数及其向量 $\mathbf{v}$, 将 c_1 和 $\mathbf{v}$ 以 c_1_v 的形式作为文件的一行记录保存到 HDFS 的 D_1 目录中, 输出所有频繁 1- 项集 L_1 .

Map 函数描述如下:

```
map (key=字节偏移, value=T) {
  for each  $c_1$  of  $T$  //  $c_1$  是候选 1-项集
    output ( $\langle c_1, \text{TID} \rangle$ );
}
```

Reduce 函数描述如下:

```
reduce(key= $c_1$ , values=list(TID)) {
  sup_count=0; //初始化  $c_1$  的支持度计数
   $\mathbf{v}[1..n]=0$ ; //初始化  $c_1$  的向量
  for each TID  $\in$  list(TID) {
    sup_count++; //  $c_1$  的支持度计数累加
     $\mathbf{v}[\text{TID}]=1$ ; //按定义 2 计算  $c_1$  的向量
  }
  将  $c_1\_v$  作为文件的一行记录保存到  $D_1$  目录;
  if (sup_count  $\geq$  minsup) //输出频繁 1-项集
    output( $\langle c_1, \text{sup\_count} \rangle$ );
}
```

(3) 主函数将矩阵 $\mathbf{A}_1$ 分解为 s 个子矩阵 $\mathbf{A}_{11} \sim \mathbf{A}_{1s}$. 具体过程是: 主函数从 D_1 目录的各个文件中读取候选 1- 项集及其向量, 对每一个候选 1- 项集 c_1 进行如下计算: 将 c_1 的向量 $\mathbf{v}$ 分解成 s 个子向量 $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_s$; 分别将 c_1 及其子量 $\mathbf{v}_i$ 以 $c_1_v_i$ 的形式作为文件 f_i 的一行记录保存到 HDFS 的 C_1 目录中.

(4) $k = 1$.

(5) 主函数将所有频繁 k - 项集 L_k 作为分布式缓存文件分发给 Map 任务节点, 将 C_k 目录中的文件 f_i 分配给 Map 任务节点.

(6) 计算频繁 $(k+1)$ 项集. Map 函数进行如下计算: 从文件 f_i 中读取候选 k - 项集 c_k 及其向量, 如果 c_k 不是频繁 k - 项集, 则删除 c_k ; 按定义 3 计算候选 $(k+1)$ - 项集 c_{k+1} 及其子向量 $\mathbf{v}_i$; 计算 c_{k+1} 的局部支持度计数, 将 c_{k+1} 及其子量 $\mathbf{v}_i$ 以 $c_{k+1_v_i}$ 的形式作为文件 f_i 的一行记录保存到 HDFS 的 C_{k+1} 目录.

Reduce 函数对 c_{k+1} 的局部支持度计数进行累加, 输出所有频繁 $(k+1)$ - 项集 L_{k+1} .

Map 函数描述如下:

```
setup() {
  读取  $L_k$ ;
   $B_k = \emptyset$ ; // 初始化动态数组  $B_k$ 
}
//map 方法删除非频繁  $k$ - 项集
map (key = 字节偏移, value =  $c_k\_v_i$ ) {
  if ( $c_k \in L_k$ )
    将  $c_k\_v_i$  添加到  $B_k$  中;
}
cleanup() {
  for each  $c_k\_v_i \in B_k$  {
     $l_1 = c_k$ ;  $\mathbf{v}'_1 = \mathbf{v}_i$ ;
    for each  $c_k\_v_i \in B_k$  {
       $l_2 = c_k$ ;  $\mathbf{v}'_2 = \mathbf{v}_i$ ;
      // 定义 3 计算  $c_{k+1}$  及其向量
      if ( $l_1[1 \dots k-1] = l_2[1 \dots k-1] \ \&\& \ (l_1[k] < l_2[k])$ ) {
         $c_{k+1} = l_1 \odot l_2$ ;
         $\mathbf{v}_i = \mathbf{v}'_1 \wedge \mathbf{v}'_2$ ;
        将  $c_{k+1} \_ v_i$  作为文件  $f_i$  的一行记录保存到  $C_{k+1}$  目录;
        loc_count = 0; // 初始化局部支持度
        for each  $a_j \in \mathbf{v}_i$  // 计算局部支持度
          loc_count +=  $a_j$ ;
        output( $\langle c_{k+1}, \text{loc\_count} \rangle$ );
      }
    }
  }
}
```

Reduce 函数描述如下:

```
reduce(key= $\leq c_{k+1}$ , values = list(loc_count)) {
  sup_count = 0; // 初始化支持度计数
  for each loc_count  $\in$  list(loc_count) {
    sup_count += loc_count;
  }
  if (sup_count  $\geq$  minsup) // 输出  $L_{k+1}$ 
    output( $\langle c_{k+1}, \text{sup\_count} \rangle$ );
}
```

(7) 如果 $L_{k+1} = \emptyset$, 则算法结束. 如果 $L_{k+1} \neq \emptyset$, 则 $k = k+1$, 重复执行 (5) ~ (7).

3.3 算法的实例分析

假设事务数据库分为 2 个分片, 从节点的个数为 2 个, 子矩阵的个数为 2 个, 最小支持度计数阈值为 2, 以图 1 所示的示例数据库说明本文算法的执行

过程. 从该示例数据库中找出所有频繁项集需要 4 个 Job, 前 3 个 Job 的执行过程如图 1 所示, 说明如下.

(1)Job1 计算所有频繁 1- 项集 L_1 , 建立矩阵 A_1 , 将 A_1 保存到 D_1 目录中. 主函数将 A_1 分解为 2 个子矩阵 A_{11} 和 A_{12} , 将 L_1 分发给从节点.

(2)Job2 的 Map 任务分别对子矩阵 A_{11} 和 A_{12} 计算局部支持度计数, 建立子矩阵 A_{21} 和 A_{22} . Job2 的 Reduce 任务对局部支持度进行累加, 输出频繁 2- 项集 L_2 .

(3) 主函数将 L_2 分发给从节点. Job3 的执行过程与是 Job2 的执行过程相同, 建立子矩阵 A_{31} 和 A_{32} , 输出频繁 3- 项集 L_3 .

(4) 主函数将 L_3 分发给从节点. 子矩阵 A_{31} 和 A_{32} 分别输入 Job4 的 Map 任务, 由于 L_3 不能连接生成候选 4- 项集, Job4 的 Reduce 任务输出的 L_4 为空, 算法结束.

对该实例的分析可以看出本文算法具有三个优点: 各节点对相同个数和规模的子矩阵计算候选 k - 项集的支持度, 实现了节点的负载平衡; 在迭代计算频繁 k - 项集的过程中, 矩阵 A_k 的规模逐步减少, 减少了算法的 I/O 时间; Map 与 Reduce 任务间的通信量较少.

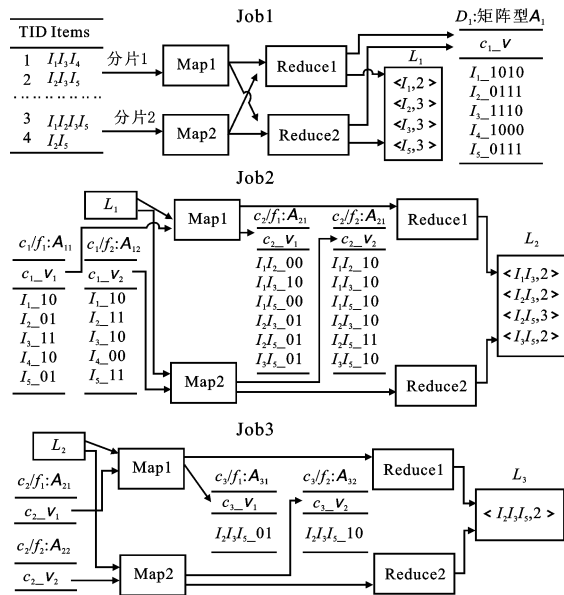


图 1 算法的执行过程示例

4 实验及结果分析

使用 8 台配置相同的计算机搭建了一个 Hadoop 集群, 其中 1 台作为主节点, 7 台作为从节点. 计算机安装的软件是 Ubuntu 12. 04 LTS、Hadoop 1. 2. 1 和 JDK 1. 7. 0_51. 使用 Java 语言实现了本文算法和 Apriori 算法, 实验的数据集 T10I4D100K、

kosarak、connect 和 pumsb_star 来自网站 <http://fimi. ua. ac. be/data/>.

4. 1 算法的执行时间测试

为了评价本文算法的执行时间, 与经典的 Apriori 算法进行了比较. 在 1 台计算机上测试了 Apriori 算法的执行时间, 分别使用 1~3 个从节点测试了本文算法的执行时间, 实验结果如表 1 所示. 从表中可以看出, 利用多个节点的计算能力缩短了挖掘频繁项集的时间.

对于稀疏型的数据集 T10I4D100K 和 kosarak, 本文算法在 1 个节点上的执行时间小于 Apriori 算法的执行时间. 由于本文算法在迭代计算频繁 k - 项集的过程中, 稀疏型数据集的矩阵规模显著减小, 从而缩短了算法的执行时间, 随着节点数的增加, 算法的执行时间进一步减少. 对于稠密型的数据集 pumsb_star 和 connect, 本文算法在 3 个节点上的执行时间小于 Apriori 算法的执行时间.

表 1 本文算法与 Apriori 算法的比较

数据集	最小支持度/ %	Apriori 算法/s	本文算法 /s		
			1 节点	2 节点	3 节点
T10I4D100K	0. 3	6 112	1 576	990	709
kosarak	0. 6	3 647	3 209	2 423	1 526
pumsb_star	35	4 225	8 416	4 842	3 336
connect	80	7 318	12 163	8 342	5 119

4. 2 算法的加速比实验

加速比的计算公式为: $S_p = t_1/t_p$, 其中, t_1 是算法在单个节点上的执行时间, t_p 是算法在 p 个节点上的执行时间^[10].

使用表 1 列出的最小支持度对 4 个数据集测试算法的加速比, 实验的结果如图 2 所示. 从图中可以看出, 本文算法对 4 个数据集都取得了较好的加速比. 由于 MapReduce 程序的主函数是串行执行的, 因此本文算法不能取得线性加速比. 随着节点数的增加, Map 任务与 Reduce 任务间的通信开销增加, 加速比性能呈下降趋势.

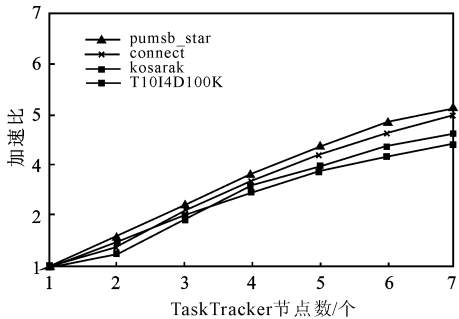


图 2 本文算法的加速比

4.3 算法的可扩展性实验

等速度可扩展性的定义^[11]:当机器规模由 P 增加到 P' ,负载规模由 W 增加到 W' ,且平均速度保持不变,则可扩展性为 $iv(P, P') = (W/P)/(W'/P')$.

将4个数据集分别复制2~7倍,使用表1给出的最小支持度阈值在这些数据集上测试算法的可扩展性,实验结果如图3所示.可以看出本文算法对稀疏型和稠密型的数据集都取得了较好的可扩展性,由于本文算法将事务矩阵分解成多个规模相同的子矩阵,各节点对相同个数的子矩阵进行计算,实现了节点的负载平衡,提高了对大数据集挖掘频繁项集的效率.随着数据规模的增加,主函数的执行时间增加,可扩展性呈下降趋势.

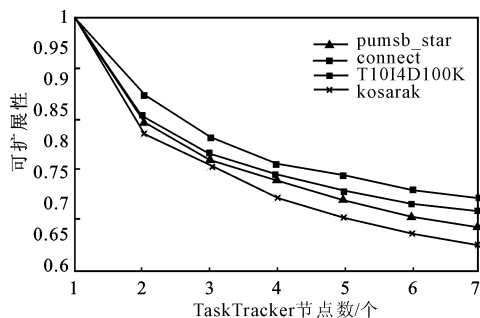


图3 本文算法的可扩展性

5 结束语

本文设计了一种基于 MapReduce 和矩阵的频繁项集挖掘算法. 实验结果表明该算法具有较好的加速比和可扩展性,适合对大数据集挖掘频繁项集.

参考文献:

[1] Agrawal R, Srikant R. Fast algorithms for mining as-
sociation rules[C]//Proceedings of the 20th Interna-

tional Conference on Very Large Data Bases. Santiago,
Chile:IEEE, 1994: 487-499.
[2] 付沙,宋丹. 基于矩阵的 Apriori 改进算法研究[J]. 微
电子学与计算机,2012,29(5):156-160.
[3] The Apache Software Foundation. Hadoop[EB/OL].
[2015-07-11] (2015-07-08). <http://hadoop.apache.org/>.
[4] Dean J, Ghemawat S. MapReduce: simplified data
processing on large clusters[J]. Communications of
the ACM, 2008, 51(1): 107-113.
[5] 张敏,姚良威,侯宇. 基于向量和矩阵的频繁项集挖掘
算法研究[J]. 计算机工程与设计, 2013, 34(3):
939-943.
[6] 李成华,张新访,金海,等. MapReduce:新型的分布式
并行计算编程模型[J]. 计算机工程与科学,2011,33
(3):129-135.
[7] 易小华,刘杰,叶丹. 面向 MapReduce 的数据处理流程
开发方法[J]. 计算机科学与探索, 2011, 5(2):
161-168.
[8] 吕婉琪,钟诚,唐印许,等. Hadoop 分布式架构下大数
据集的并行挖掘[J]. 计算机技术与发展,2014, 24
(1):22-25.
[9] 施亮,钱雪忠. 基于 Hadoop 的并行 FP-Growth 算法的
研究与实现[J]. 微电子学与计算机, 2015, 32(4):
150-154.
[10] 陈兴蜀,张帅,童浩,等. 基于布尔矩阵和 MapReduce
的 FP-Growth 算法[J]. 华南理工大学学报(自然科学
版), 2014, 42(1): 135-141.
[11] 王与力,杨晓东. 一种更有效的并行系统可扩展性模型
[J]. 计算机学报,2001,24(1):84-90.

作者简介:

周国军 男,(1975-),硕士,讲师. 研究方向为数据挖掘、云
计算. E-mail:ylsyzgj@126.com.
龚榆桐 男,(1982-),硕士,副教授. 研究方向为计算机应
用、电子商务.

(上接第 118 页)

[4] 宁爱平,张雪英,刘俊芳. ABC-PSO 算法优化混合核
SVM 参数及应用[J]. 数学的实践与认识,2014, 44
(18):158-165.
[5] 孙轶轩,邵春福,岳昊,等. 基于 SVM 灵敏度的城市交
通事故严重程度影响因素分析[J]. 吉林大学学报(工
学版),2014,44(5):1315-1320.
[6] 刘海红,杜英举,姚伟华. 不同意识脑电的双谱切片特
征分析与分类[J]. 电子设计工程,2015,23(8):42-45.
[7] 曹健,孙世宇,段修生,等. 基于 KKT 条件的 SVM 增
量学习算法[J]. 火力与指挥控制, 2014, 39(7):
139-143.
[9] 张保华,黄文倩,李江波,等. 基于 I-RELIEF 和 SVM

的畸形马铃薯在线分选[J]. 吉林大学学报:工学版,
2014,44(6):1811-1817.
[9] 宋航,张忠林,谢冰. 基于 SVM 分类器的动态关联规则趋
势度挖掘方法[J]. 信息通信, 2014,27(6):56-57.
[10] 张翔,陈林,基于果蝇优化算法的支持向量机故障诊断
[J]. 电子设计工程,2013,21(16):90-93.

作者简介:

郑金芳 女,(1979-),硕士研究生,副教授. 研究方向为计
算机软件和数据库.
刘远超(通讯作者) 男,(1983-),硕士研究生,讲师. 研究方
向为计算机应用. E-mail:jsjxlyc@163.com.