R-means：以关联规则为簇中心的文本聚类*)

龙 昊 冯剑琳 李 曲

(华中科技大学计算机科学与技术系 武汉 430074)

摘 要 本文将k-means与关联规则(或频繁项目集)相结合,提出了一种新的文本聚类算法R-means。R-means算法以关联规则作为簇中心,通过类似于k-means的迭代优化得到最终的簇。因此R-means不仅继承了k-means的简单性,而且用关联规则产生的簇描述易于为人们所理解。在几个实际数据集上的实验表明该算法可以得到高精度和高性能。

关键词 关联规则,频繁项目集,簇中心,关联文本聚类

R-means: Exploiting Association Rules as Means for Text Clustering

LONG Hao FENG Jian-Lin LI Qu

(Department of Computer Science, Huazhong University of Science and Technology, Wuhan 430074)

Abstract This paper proposes a new text clustering algorithm called R-means which integrates k-means with association rule (or frequent itemset). R-means exploits association rules as means of clusters and refines clusters by an iterative procedure which is similar to that of k-means. R-means not only inherits the simplicity of k-means, but also generates more comprehensive cluster labels which are described by association rules. The experiments with several real data sets have demonstrated that R-means can achieve quite well precision and high performance.

Keywords Association rules, Frequent itemset, Means of clusters, Associative text clustering

1 引言

文本聚类(Text Clustering)是一种无指导的机器学习方法,它广泛应用于信息检索、决策支持等领域。M. Steinbach在文[7]指出,在传统算法如UPGMA[5],k-means[5]等中,以k-means算法的一个变体bisecting k-means效果最好。k-means算法最初由MacQueen于1967年提出[3],它通过迭代地重分配来得到局部优化的聚类。Selim和Ismail于1984年证明了k-means的收敛性[5]。由于算法效率较高并且容易实现,k-means在工业界得到广泛应用,但是k-means未能解决文本聚类所面临的高维问题。

关联聚类算法(主要是HFTC[4]和FIHC[1])则是新近提出的聚类方法,它利用频繁项集(frequent itemset)来表示文本,从而大幅度降低了文本的维度。这里频繁项集是指在不低于某一比例的文档中出现的单词集合[1,4]。HFTC算法每次选择下一个较频繁的项集(itemset)来构造下一个簇,选择的标准是这个频繁项集覆盖的文档集与其它频繁项集覆盖的文档集交集最小。这种聚类方法依赖于频繁项集的选择顺序。FIHC算法则不需要按顺序选择频繁项集来生成各个簇。它进行聚类的标准是:属于同一个簇C的文档之所以相似是因为它们共享了较多的频繁项集,而不同的簇则应该尽可能少地共享频繁项集。根据这一标准,FIHC采用了相应的文档相似度度量生成聚类,并利用频繁项集之间的包含关系来生成层次。

本文将k-means与利用频繁项集来表示文本相结合,提出一种新的平面型(即非层次型)文本聚类算法,称为R-means(Rules as means)。该算法采用k-means的聚类过程,但是融入关联聚类的观点,利用频繁项集构成的规则集作为簇中心,并且采用与FIHC不同的文档相似度来对文本进行分配。实验证明,R-means在具有k-means简单、易实现的优点的同时,也能够获得可与FIHC媲美的高精度和高性能。

本文第2节将详细描述R-means算法;实验结果和比较将放在第3节说明;最后对全文进行总结。

2 R-means

R-means的整体过程可如下描述:

1. 利用频繁项集构建初始簇;
2. 为每个簇构建簇规则集;
3. 利用簇规则集重分配文本;
4. 迭代2、3两步,直到文本集分布不再发生变化或者到达指定迭代次数。

下面几个小节将利用一个运行实例详细介绍以上各个步骤。

2.1 构建初始聚类

R-means利用文本集的全局频繁项集来构造初始簇,全局频繁项集合是这样的一个项集:

假设D表示待聚类文本集。给定一个项t,也就是一个单词,如果它在不低于某个指定比例的文档中出现,称t为一个全局频繁项。这个指定的比例称为全局最小支持度,记为$gminsup$。D中的所有频繁项就构成全局频繁项集合,记为F。

得到全局频繁项集合后,对每个全局频繁项,构造一个初

*)本文受国家自然科学基金(编号60303030)和重庆自然科学基金(编号8721)支持。龙 昊 硕士研究生,研究方向:文本挖掘。冯剑琳 博士,研究方向:文本挖掘和联机分析处理。李 曲 博士研究生,研究方向:文本挖掘和联机分析处理。

始簇，这个初始簇是由所有包含该项的文档构成的。

例：这里借用文[1]中提供的一个例子文档集来说明 R-means 算法。如表 1 所示，这个文档集由从数据集 *classic4*[13] 中选取的 12 个文档组成。为简单起见，这里直接采用单词的频度作为文档向量的权值。表 2 给出了这个文档集的全局频繁项，表 3 则描述了构造的初始簇。

表 1　文本集

文档	文档向量					
	(flow,	form,	layer,	patient,	result,	treatment)
cisi. 1	(0,	1,	0,	0,	0,	0)
cran. 1	(1,	1,	1,	0,	0,	0)
cran. 2	(2,	0,	1,	0,	0,	0)
cran. 3	(2,	1,	2,	0,	3,	0)
cran. 4	(2,	0,	3,	0,	0,	0)
cran. 5	(1,	0,	2,	0,	0,	0)
med. 1	(0,	0,	0,	8,	1,	2)
med. 2	(0,	1,	0,	4,	3,	1)
med. 3	(0,	0,	0,	3,	0,	2)
med. 4	(0,	0,	0,	6,	3,	3)
med. 5	(0,	1,	0,	4,	0,	0)
med. 6	(0,	0,	0,	9,	1,	1)

表 2　全局频繁项集 (gminsup=35%)

全局频繁项	支持度
flow	42%
form	42%
layer	42%
patient	50%
result	42%
treatment	42%

表 3　初始簇

簇	簇文档集
C(flow)	cran. 1, cran. 2, cran. 3, cran. 4, cran. 5
C(form)	cisi. 1, cran. 1, cran. 3, med. 2, med. 5
C(layer)	cran. 1, cran. 2, cran. 3, cran. 4, cran. 5
C(patient)	med. 1, med. 2, med. 3, med. 4, med. 5, med. 6
C(result)	cran. 3, med. 1, med. 3, med. 4, med. 6
C(treatment)	med. 1, med. 2, med. 3, med. 4, med. 6

2.2　构建簇规则集

这一步将选用各个簇的簇频繁项集合作为簇规则集，其中一个簇频繁项对应一条规则。在这里，簇频繁项可以这样描述：

一个簇频繁项首先也是一个全局频繁项，并且在不低于指定比例的簇文本中出现。这个比例称为最小簇支持度，记为 *cminsup*。一个簇中的所有簇频繁项构成该簇的簇频繁项集。

选择簇频繁项作为规则是基于以下考虑：簇频繁项反映了该簇文档的共性，可以用这种共性来描述文本之间的相似性。这一考虑与 *FIHC* 类似。

规则集中的每个规则 R 都具有两个属性：支持度和置信度，分别记为 $R.sup$ 和 $R.conf$，它们的计算方法如下：

给定一个簇 C，设 $cov(C,R)$ 表示簇 C 的文档中出现规则 R 的所有文档，$|C|$ 表示簇 C 中的文档总数，$cov(R)$ 表示全部文档中出现 R 的文档集合，则

$$R.sup=|cov(C,R)|/|C|,$$

$$R.conf=|cov(C,R)|/|cov(R)|$$

$R.sup$ 量化了规则 R 的普遍性，而 $R.conf$ 则量化了规则 R 的可信性。

在为某个簇生成规则集时，首先将该簇包含的所有全局频繁项集作为候选规则集，然后计算出每个候选规则的支持度和置信度，最后修剪掉所有支持度小于指定 *cminsup* 的规则。

例：对于簇 $C(flow)$ 的两个候选规则 $R(flow)$ 和 $R(form)$，可以计算出它们的置信度和支持度如下：

$$R(flow).sup=cov(flow,C(flow))/|C(flow)|=5/5=1.0$$

$$R(flow).conf=cov(flow,C(flow))/|cov(flow)|=5/5=1.0$$

$$R(form).sup=cov(form,C(flow))/|C(flow)|=2/5=0.4$$

$$R(form).conf=cov(form,C(flow))/|cov(form)|=2/5=0.4$$

如果我们设置 *cminsup* 为 50%，那么 $R(form)$ 将被修剪掉。表 4 给出了由初始簇构造出来的簇规则集。

2.3　重分配文本

这一步将根据上一步生成的簇规则集来优化聚类，通过计算每个文本与各个簇的匹配程度，将它分配到与它最匹配的一个或多个簇中。给定簇 S，设它的簇规则集为 $CL(S)$，我们用文档 d 内包含的所有规则的置信度的加权和来衡量文档 d 与簇 S 的匹配程度，即：

表 4　初始簇的簇规则集(cminsup=50%)

簇	簇规则集
C(flow)	{flow, sup=1.0, conf=1.0}, {layer, sup=1.0, conf=1.0}
C(form)	{form, sup=1.0, conf=1.0}
C(layer)	{flow, sup=1.0, conf=1.0}, {layer, sup=1.0, conf=1.0}
C(patient)	{patient, sup=1.0, conf=1.0}, {result, sup= 0.67, conf=0.8}, {treatment, sup=0.83, conf=1.0}
C(result)	{patient, sup=0.8, conf=0.67}, {result, sup=1.0, conf=1.0} {treatment, sup=0.8, conf=0.8}
C(treatment)	{patient, sup=1.0, conf=0.83}, {result, sup=0.8, conf=0.8} {treatment, sup=1.0, conf=1.0}

$$Score(S\leftarrow d)=\sum_{x} n(x)*R(x).conf, x\in CL(S)\cap d$$

其中 x 表示文档 d 与 $CL(S)$ 的交集中的项，$n(x)$ 表示该项在文档 d 的文档向量中的权值，$R(x)$ 表示 $CL(S)$ 中 x 对应的规则。由于规则的置信度反映的是规则所在的文档属于该簇的概率，因此 $Score(S\leftarrow d)$ 能够在一定程度上反映文档 d 与簇 S 的匹配程度，并且值越大，匹配程度越高。

例：计算文档 med. 2 与各个簇的匹配得分：

$$Score(C(flow)\leftarrow med.2)=0$$

$$Score(C(form)\leftarrow med.2)=1*1.0=1.0$$

$$Score(C(layer)\leftarrow med.2)=0$$

$$Score(C(patient)\leftarrow med.2)=4*1.0+3*0.8+1*1.0=7.4$$

$Score(C(result) \leftarrow med.2) = 4*0.67+3*1.0+1*0.8 = 6.1$

$Score(C(treatment) \leftarrow med.2) = 4*0.83+3*0.8+1*1.0 = 6.7$

直觉上，应该将文本分配到与它匹配程度最高的簇当中去。但是由于当前的簇并不稳定，也把这个文本同时分配到与最高匹配得分相近的簇中去。这里引入一个参数称为分配因子，记为$\sigma(0<\sigma\leqslant 1)$，来实现这一过程。如果文档与某个簇的匹配得分大于最高匹配得分与σ的乘积，那么该文档就被分配到这个簇中。

例：文档 med.2 的分配过程。med.2 与 C(patient)得到匹配的最高得分为 7.4，设定σ为 0.8，那么 med.2 与 $C(patient)$，$C(result)$ 和 $C(treatment)$的匹配得分都大于 $7.4*0.8=5.92$，因此 med.2 将被分配到这三个簇中。表 5 给出了经过这次重分配后的所有簇。

表 5 经过一次重分配后的聚类(σ=0.8)

簇	簇文档集
C(flow)	cran.1, cran.2, cran.3, cran.4, cran.5
C(form)	cisi.1
C(layer)	cran.1, cran.2, cran.3, cran.4, cran.5
C(patient)	med.1, med.2, med.3, med.4, med.5, med.6
C(result)	med.1, med.4
C(treatment)	med.1, med.2, med.3, med.4, med.5, med.6

2.4 迭代过程

迭代 2、3 两步，直到文本集分布不再发生变化或者到达指定迭代次数。

另外，在每一次迭代结束后，那些不包含文本的空簇将要删除。如果多个簇包含相同的文本集，那么只需保留其中一个簇。

3 实验评估

在本节，将详细介绍实验设置以及与其他几个算法的比较情况。参与比较的算法包括 UPGMA [5]、bisecting k-means [7]、HFTC [4]和 FIHC [1]。为了得到这几个算法的实验数据，本文利用了 CLUTO-2.0 Clustering Toolkit [6]，并且参考了文[1]中提供的部分结果。

3.1 实验数据集与预处理

我们选用了五个实际数据集来进行实验评估，这五个数据集是 *Classic4*、*Hitech*、*Wap*、*Reuters* 和 *Re0*。*Classic4* 包括 7094 篇文档，这些文档被分为四个类：CACM、CISI、CRAN 和 MED[13]。Hitech 则由从 San Jose Mercury 报纸中摘取的一部分组成[9]。*Wap* 包含了从 Yahoo! 中抽取的 1560 个网页[10]。*Reuters* 和 *Re0* 则来自路透社的新闻报道[11]。对于 Reuters 数据集来说，只选择了其中仅属于一个类的文档。表 6 给出了以上几个数据集的概况。

表 6 测试数据集

数据集	文档数	类数目	类大小	词汇数
Classic4	7094	4	1033-3203	12009
Hitech	2301	6	116-603	13170
Re0	1504	13	11-608	2886
Reuters	8654	65	1-3725	16641
Wap	1560	20	5-341	8460

文本预处理包括去停用词(stopword removal)、统一词形(word stemming)等处理。另外每个文本用一个文本向量描述，除了 HFTC，其它的算法均采用 tf×idf[12]方法来描述文本向量每个维的权重。HFTC 只考虑单词的出现，而不关心其出现频率。

3.2 评估方法

在所有这几种算法中，HFTC 和 FIHC 是层次型的聚类算法，而其余几种，包括 R-means，都是平面型的聚类算法。我们选用比较普遍的 F-Measure[7,1]值来衡量层次型和平面型的算法。设 $C=\{C_1, C_2, \cdots, C_m\}$是生成的簇集，$K=\{K_1, K_2, \cdots, K_n\}$是手工分类结果。特别地，对于 C_i 和 K_j 来说，查全率 $R(C_i, K_j)$和查准率 $P(C_i, K_j)$定义为：

$$R(K_i,C_j)=|K_i \cap C_j|/|K_i|,$$

$$P(K_i,C_j)=|K_i \cap C_j|/|C_j|$$

C_i 和 K_j 的 F-Measure 值则定义为：

$$F(K_i,C_j)=\frac{2*R(K_i,C_j)*P(K_i,C_j)}{R(K_i,C_j)+P(K_i,C_j)}$$

F-Measure 值越高，说明 C_i 和 K_j 越相似。对于一个手工分类，选择与之最相似的簇作为匹配簇。而整个簇集的整体 F-Measure 值则是所有手工分类与其匹配簇间的 F-Measure 值的加权平均值，即

$$F(C)=\sum_{k_i \in K}\frac{|K_i|}{|D|}*\max_{C_i \in C}\{F(K_i,C_j)\}$$

$F(C)$越大，表明整体聚类精度越高。

3.3 实验结果

在本小节，首先介绍 R-means 算法的参数选择，然后提供 R-means 算法与其他算法的比较情况。

参数选择：R-means 算法共有三个参数需要确定，分别是最小全局支持度(*gminsup*)、最小簇支持度(*cminsup*)和分配因子(σ)。

1. *最小全局支持度(gminsup)* *gminsup* 用于全局频繁项集的挖掘，参见 2.1 节。大量的实验表明：当文档数量在 1000～10000 之间的时候，*gminsup* 应设置为 2%～10%，文档数量越少，*gminsup* 值应越大。图 1 显示了聚类精度随 *gminsup* 变化的情况。

2. *簇最小支持度(cminsup)* *cminsup* 用于候选规则的修剪，参见 2.2 节。经验表明，*cminsup* 设置为 0.25～0.50 之间的值是比较合适的，簇越小，*cminsup* 值应越大。*cminsup* 的值可利用经验公式由程序实现。图 2 给出了聚类精度随 *cminsup* 值变化的情况。

3. *分配因子(σ)* 分配因子用于文本分配，参见 2.3 节。多次实验表明，σ 设置为 0.70～0.85 之间的值比较合适。

另外，在进行的几组实验中我们没有指定迭代次数，而是由程序自动运行至簇不再变化时停止。

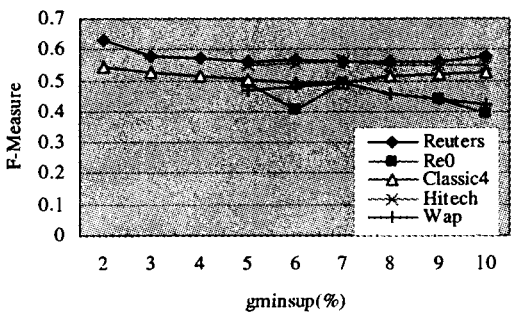

图 1 全局最小支持度的影响

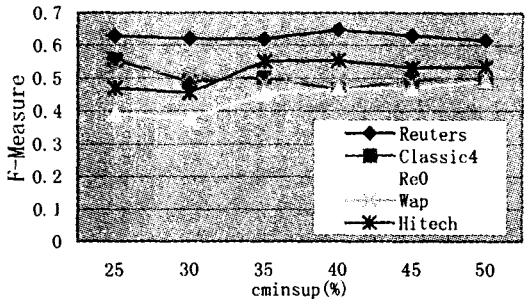

图 2 簇最小支持度的影响

实验比较：表 7 给出了算法平均精度的比较结果，表 8 则是算法最高精度的比较结果。

R-means 的平均精度的获取是通过固定 *gminsup* 为一个合适的值，然后通过变化一组其余参数，最后将得到的 F-Measure 值取平均得到。而最高精度则是一组实验中获得的最高 F-Measure 值。

表 7 平均精度比较

数据集	整体 F-Measure(平均)				
	R-means	FIHC	UPGMA	Bi-kmeans	HFTC
Classic4	0.50	0.54	—	0.44	0.61
Hitech	0.52	0.42	0.38	0.37	0.37
Re0	0.44	0.45	0.40	0.34	0.43
Reuters	0.63	0.60	—	0.39	0.49
Wap	0.50	0.52	0.51	.45	0.35

表 8 最高精度比较

数据集	整体 F-Measure(平均)				
	R-means	FIHC	UPGMA	Bi-kmeans	HFTC
Classic4	0.56	0.62	—	0.59	0.61
Hitech	0.55	0.45	0.47	0.54	0.37
Re0	0.49	0.53	0.47	0.38	0.43
Reuters	0.65	0.61	—	0.48	0.49
Wap	0.53	0.57	0.59	0.57	0.35

由表 7 和表 8 可以看出，*R*-means 无论是平均精度还是最高精度总是能达到或接近最优结果。

3.4 性能

为测试算法性能，我们选择五个数据集中最大的数据集 *Reuters* 来进行该项测试，测试平台是系统为 Windows2000 的 Celeron 2G 的 PC 机。当 gminsup 设置为 10%时，一组实验表明，尽管该数据集包含超过 8000 个文档，算法的平均运行时间仅为 53 秒，并且算法所需的平均内存量也仅为 5.4 兆。

结论 本文将传统 *k*-means 算法和关联文本聚类技术相结合，提出了一种新颖的文本聚类方法。通过在多个现实数据集上的实验证明，该算法可以取得很高的精度，并具有良好的性能。

参 考 文 献

1 Fung B C M, Wang K, Ester M. Hierarchical Document Clustering Using Frequent Itemsets. In: Proc. of the 2003 SIAM Intl. Conf. on Data Mining(SIAM'03)
2 Agrawal R, Srikant R. Fast algorithms for mining association rules. VLDB-94, 1994
3 MacQueen. Some methods for classification and analysis of multivariate observations. In: Proc. 5th Berkeley Symp., I, 1967. 281～297
4 Beil F, Ester M, Xu X. Frequent term-based text clustering. In: Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD)'2002, Edmonton, Alberta, Canada, 2002
5 Dubes R C, Jain A K. Algorithms for Clustering Data. Prentice Hall College Div, Englewood Cliffs, NJ, March 1998
6 Karypis G. Cluto 2.0 clustering toolkit. April 2002. http://www-users.cs.umn.edu/~karypis/cluto/
7 Steinbach M, Karypis G, Kumar V. A comparison of document clustering techniques. In: KDD Workshop on Text Mining'00, 2000
8 Mitchell T. Machine Learning. McGraw Hill, 1997
9 Text REtrival Conference TIPSTER, 1999. http://trec.nist.gov/
10 Han E-H, Boly D, Gini M, et al. WebACE: A Web Agent for Document Categorization and Exploration. In: Proc. Agents 98
11 http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html
12 van Rijsbergen C J. Information Retrieval. Dept. of Computer Science, University of Glasgow, Butterworth, London, 2 edition, 1979
13 Classic. ftp://ftp.cs.cornell.edu/pub/smart/

(上接第 90 页)

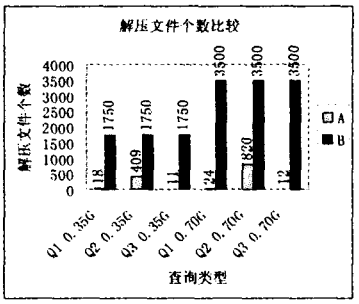

图 4 查询算法 A 与 B 解压文件个数比较(SIZE=200kb)

参 考 文 献

1 Luo J Z, Li J Z, Wang H Z, et al. The Compression of Massive Offline Relations. WAIM, 2004. 634～639
2 Goldstein J, Ramakrishnan R, Shaft U. Compressing Relations and Indexes. ICDE, 1998. 370～379
3 Westmann T, Kossmann D, Helmer S, et al. The Implementation and Performance of Compressed Databases. SIGMOD, 2000, 29(3): 55～67
4 张艳秋，李建中. 海量信息存储系统中一种基于磁带的文件系统. 高技术通讯，2003，13(2)
5 Nelson M, Gailly J-l. The Data Compression 2nd Edition. New York: M&T Books, 1995
6 Poess M, Potapo D. Data compression in oracle. VLDB, 2003. 937～947
7 Furtado P, Madeira H. FCompress: A New Technique for Queriable Compression of Facts and Datacubes. IDEAS, 2000. 197～206
8 Gao H, Li J. Cube Algorithms on Very Large Compressed Data Warehouses. Journal of Software, 2001, 12(6)
9 Li J Z, Srivastava J. Efficient Aggregation Algorithms for Compressed Data Warehouses. TKDE, 2002, 14(3): 515～529
10 Margaritis D, Faloutsos C, Thruns S. NetCube: A Scalable Tool for Fast Data Mining and Compression. VLDB, 2001. 311～320
11 Transaction Processing Performance Council. TPC BENCHMARK H (Decision Support) StandardSpecification. http://www.tpc.org/tpch/default.asp
12 Li J, Rotem D, Srivastava J. Aggregation Algorithms for Very Large Compressed Data Warehouses. VLDB, 1999. 651～662