Incremental Document Clustering for Web Page Classification

Wai-chiu Wong and Ada Wai-chee Fu

Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong

Summary. Motivated by the benefits in organizing the documents in Web search engines, we consider the problem of automatic Web page classification. We employ the clustering techniques. Each document is represented by a feature vector. By analyzing the clusters formed by these vectors, we can assign the documents within the same cluster to the same class automatically. Our contributions are the following: (1) We propose a feature extraction mechanism which is more suitable to Web page classification. (2) We introduce a tree structure called the DC-tree to make the clustering process incremental and less sensitive to the document insertion order. (3) We show with experiments on a set of Internet documents from *Yahoo!* that the proposed clustering algorithm can classify Web pages effectively.

1 Introduction

We consider document clustering for Web pages. Traditionally, the document classification task is carried out manually. In order to assign a document to an appropriate class, people would analyze the contents of the document first. Therefore a large amount of human effort would be required. There has been some research work conducted on automatic text classification. One approach is to learn the text classifiers by using the machine learning techniques. However, these algorithms are based on a set of positive and negative training examples for learning the text classifiers. The quality of the resulting classifiers highly depends on the fitness of the training examples. There are many terms and classes in the World Wide Web (or just the Web), and many new terms and concepts are created everyday. It is quite impossible to have domain experts to identify training examples to learn a classifier for each text class in the above manner.

In order to make the document classification process automatic, clustering techniques have been employed. The attractiveness of cluster analysis is that it can find clusters directly from the given data, without relying on any pre-determined information such as training examples provided by domain experts. A document is commonly represented by a feature vector. Typically, each feature corresponds to a key word or phrase appearing in the set of documents. Each entry of the vector stores a numeric weight for the corresponding feature of the document. After extracting the feature vectors of the documents, we can apply the clustering algorithm on this set of the vectors as in conventional high dimensional data clustering. The resulting document clusters together with the representative features (i.e., the key words or phrases with enough *document support* within the cluster) are thus reported to the user.

In this paper, we introduce a tree structure called the **DC-tree** (Document-Clustering Tree) which can cluster documents without any training set. With the DC-tree, an incoming data object is not forced to be inserted into the lower level when there does not exist a child node that is similar enough to this data object. This prevents some dissimilar data from being put together. As a result, the DC-tree-based clustering algorithm is less sensitive to the document insertion order and is more tolerant towards noise documents.

We believe that our method can be useful in a number of ways: (1) For preprocessing on Web page classification so that users can choose a suitable class before searching, this helps the search to be more focused and efficient. (2) For online classification, so that when a large number of results are returned from a search, the technique can classify the results and provide better guidance to user for further searching. (3) For incremental Web page classifications after updating on the repository.

2 Related Work

BIRCH [6] is a dynamical and incremental method to cluster the incoming points. An important idea of BIRCH is to summarize a cluster of points into a clustering feature vector. This summary uses much less storage than storing all data points in the cluster. A CF-tree is built which splits dynamically. Clusters are stored in the leaf nodes. In this way, the memory requirement problem of K-Means is resolved and thus the method is suitable for large data sets.

In one method presented in [5], a B^+-tree-based structure is used to index the document feature vectors. For the purpose of clustering, each leaf node is a document and each nonleaf node n is the representative of the cluster containing the documents in the leaf nodes of the subtree rooted at n. Similar to the CF-tree and the B^+-tree based method, this is a balanced tree, and has the problem of possibly forcing non-similar items into the same node or cluster.

3 Feature Extraction and Document Clustering

Our first task is to identify a good feature extracting method suitable for the Web environment. In this section, we propose such a method. The document and document cluster representation will also be described. Finally, the clustering quality evaluation method will be presented.

3.1 Document Feature Extraction

We propose a feature extracting method for the Web document clustering algorithm which does not depend on the term frequency. This method balances the trade-off between the coverage and the number of features used for document representation. In our problem domain the clustering aims to aid in information retrieval in Web searching by narrowing down the search scope. In such a scenario, the user may not want too many clusters in the result. Also, very large clusters or very small clusters are not desirable. Very large clusters cannot help to narrow the search scope. Very small clusters can increase the total number of clusters, and it may actually be caused by noise. The parameter k is used to set an *approximate* number on the cluster size. Hence the number of clusters is approximately N/k, where N is the total number of documents. The proposed method involves the following steps:

1. Randomly select a subset of documents with size m from the corpus.
2. Extract the set of words that appear at least once in the documents. Remove stop words and combine the words with the same root by using the stemming technique.
3. Count the document frequency of the words which are extracted in Step 2.
4. Set $lower = k$ and $upper = k$
5. Select all words with document frequency in the range from $lower$ to $upper$
6. Check if the *coverage* of these words is larger the pre-defined threshold. If so, stop. If no, set $lower = lower - 1$ and $upper = upper + 1$, and then goto Step 5.

In order to extract the representative features from the documents, we randomly select a set of sample documents for feature extraction in Step 1. We have carried out some experiments (see [4]) which show that this feature extraction method can extract a set of good features for Web document clustering. A stop-word list is used to remove the meaningless words. Stemming technique is used to combine those words in similar form.

Since shorter feature vectors lead to shorter clustering time, Steps 4 to 6 try to minimize the number of features and obtain reasonable coverage for the features. Assume the user wants the resulting cluster to contain about k documents. In the ideal case, a feature for a cluster will appear only in the cluster and hence the document frequency of the feature is k. Therefore, we first select the features with document frequency equal to k, by setting *lower* and *upper* to k in Step 4. The range { *lower, upper* } is enlarged repeatedly in Step 6 to ensure sufficient coverage for the resulting feature set. We see that N/k is only a rough guideline, the actual number of clusters of the clustering result may not be the same as N/k. The method also makes use of a **coverage threshold** to ensure that the features selected has sufficient coverage. In our experiments (see [4]), we find that 0.8 is a good coverage threshold value.

3.2 Document Representation

In our algorithm, a document (D_i) is represented in the following form: $D_i = (W_i, ID_i)$, where ID_i is the document identifier which can be used to retrieve document (D_i) and W_i is the feature vector of the document: $W_i = (w_{i1}, w_{i2}, \ldots, w_{in})$. Here n is the number of extracted features, and w_{ij} is the weight of the j-th feature, where $j \in \{1, 2, ..., n\}$. In our algorithm, a binary weighting scheme has been used. That is, w_{ij} is equal to 1 if D_i contains the j-th feature. Otherwise, w_{ij} is equal to 0. As mentioned in Section 3.1, a Web page typically does not contain many words so that the term frequency of a word cannot indicate the actual importance of this word. Therefore, the binary weighting scheme is more suitable to our problem domain.

3.3 Document Cluster (DC)

A **Document Cluster** value (DC) is a triple storing the information that we maintain about a set of documents within the same cluster: (1) the number of documents, (2) the set of document identifiers, and (3) the feature vector of the cluster.

Definition 1. (DC) Given N documents in a cluster: $\{D_i, D_2, \ldots, D_N\}$, the Document Cluster (DC) entry of a node is defined as a triple: DC = (N,ID,W), where N is the number of documents in the cluster, ID is the set of the document identifiers of the documents in the cluster, i.e. $ID = \{ID_1, ID_2, \ldots, D_N\}$, and W is the feature vector of the document cluster, i.e. $W = (w_1, w_2, \ldots, w_n)$, where $w_j = \sum_{i=1}^{N} w_{ij}$, and n is the number of extracted features.

This triple not only summarizes the document frequency inside the cluster, but also can be used to measure the closeness between two clusters. The following lemma gives us the flexibility to combine two clusters into one and gives a DC value for the combined cluster.

Lemma 1. *(Additivity) Let $DC_1 = (N_1, ID_1, W_1)$, and $DC_2 = (N_2, ID_2, W_2)$ be the DC entries of two disjoint document clusters, where disjoint means a document does not belong to more than one cluster at the same time. Then the new DC entry, DC_{new}, of the cluster that is formed by merging the two disjoint clusters, is: $DC_{new} = (N_1 + N_2, ID_1 \cup ID_2, W_1 + W_2)$, where $W_1 + W_2 = (w_{11} + w_{21}, w_{12} + w_{22}, \ldots, w_{1n} + w_{2n})$, and n is the number of extracted features.*

3.4 Evaluation Techniques

In order to evaluate the quality of the clustering result, we adopt the *F-Measure* evaluation technique which was first introduced in [3]. The details of this evaluation methodology is described below.

For each hand-labeled topic T in the document set, suppose that a cluster X corresponding to that topic is formed.

N_1 = no. of documents of topic T in cluster X
N_2 = no. of documents in cluster X
N_3 = total no. of documents of topic T
$P = \text{Precision}(X, T) = \frac{N_1}{N_2}$
$R = \text{Recall}(X, T) = \frac{N_1}{N_3}$

The F-measure [3] for the topic T is defined as below:

$$F(T) = \frac{2PR}{P+R}$$

With respect to a topic T, we consider the cluster with the highest F-Measure to be the cluster C for T, and that F-Measure becomes the score for topic T. The overall F-Measure [2] for the tree clustering result is the weighted average of the F-Measure for each topic T:

$$Overall\_F\_Measure = \frac{\sum_{T \in M} (|T| \times F(T))}{\sum_{T \in M} |T|}$$

where M is the set of topics, $|T|$ is the number of documents of topic T, and $F(T)$ is the F-Measure for topic T.

4 DC-tree

In this section we propose a Web document clustering algorithm by means of a Document Cluster tree (DC-tree). In the DC-tree, each node can be considered as a document cluster. The tree structure is used to guide the incoming document object to an appropriate Document Cluster (DC) at the leaf nodes. It is similar to the B^+-tree [5] in that the index records in its leaf nodes contain pointers to data objects, but it is not a height-balanced tree. This structure is designed so that assigning a document into a cluster requires visiting only a small number of nodes.

A **DC-tree** is a tree with four parameters: branching factor (B), two similarity thresholds (S_1 and S_2, where $0 \leq S_1, S_2 \leq 1$) and the minimum number of children of a node (M). A non-leaf node contains at most B entries of the form $(DC_i, Child_i)$, where $i = 1, 2, \ldots, B$, "$Child_i$" is a pointer to its i-th child node or a document, and DC_i is the DC entry of the sub-cluster represented by its i-th child or a document. So, a non-leaf node represents a cluster that is made up of all the sub-clusters represented by its entries. A DC leaf node contains at most B entries, each is an entry of the form (DC_i, Doc_i), where $i \in \{1, 2, \ldots, B\}$, "$Doc_i$" is a pointer to a document or a set of documents, and DC_i is the DC entry of the corresponding sub-cluster. We call the set of documents under such a pointer a **document leaf node**, to distinguish it from the tree leaf node or **DC leaf node** (see Figure 1). A DC leaf node also represents a cluster made up of all the sub-clusters represented by its DC entries. The DC-tree allows an incoming document entry to be inserted in a new document leaf node at different levels of the tree. So, the DC-tree is not a height balanced tree. Figure 1 shows a sample DC-tree with a height of 2, $B = 3$, $M = 2$. Note that the tree is not balanced. In the tree construction, two thresholds are used:

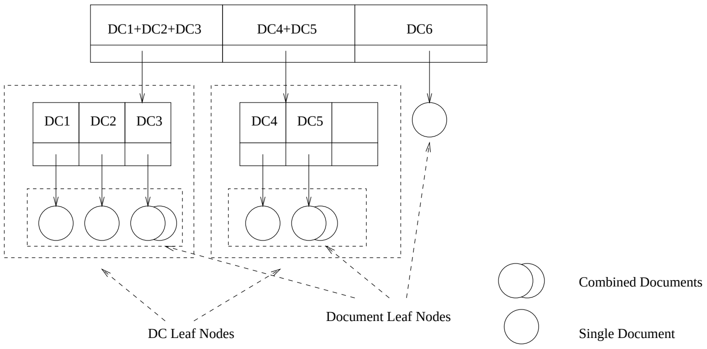

Fig. 1. Example of a DC-tree

1. **Threshold** S_1: In order to prevent poor document clustering result (i.e. documents in different classes are assigned to the same subtree or cluster) caused by document insertion order, S_1 is used to determine whether the incoming document entry E can be passed into the next level during the insertion process. If there exists an child entry of the current node such that the similarity value between this entry and the incoming entry is higher than the S_1, the incoming entry can be passed to the corresponding child node. Otherwise, the incoming entry will be added to the current node as a new document leaf node.
2. **Threshold** S_2: Since the DC-tree is used for document clustering but not indexing, it is not necessary to force each leaf entry to point to a single document. In order to reduce the insertion time, the incoming document entry can be **combined** with a leaf entry, if their similarity value is higher than S_2. This entry combination makes use of the merging as described in Lemma 1, it can save on node insertion and splitting operations so that the insertion time can be reduced.

The DC-tree is a compact representation of the dataset because each entry in a leaf node is not a single data point but a cluster of data points (the leaf node absorbs as many data points as the threshold S_2 allows). With the above DC-tree definition, the tree size will be a function of the threshold values S_1 and S_2. In general, the tree height increases with S_1 and the tree size increases with S_2. If we set S_1 to zero and S_2 to any number larger than one, the DC-tree will be similar to a balanced tree like a B^+-tree or an R-tree. The data deletion and merging algorithm is similar to that of the B^+-tree.

4.1 Identifying the Interesting Clusters

The cluster identifying process starts from the root of the tree. A breath-first-search algorithm is applied to discover the interesting clusters. An interesting cluster is defined as a cluster that contains representative features and whose size is in a predetermined range. We can make use of the *lower* and *upper* values found in our proposed feature extracting method to determine the cluster size range. Let l and u be the lower bound and upper bound of the cluster size range, then l and u can be determined by the following equations: (1) $l = lower \times \frac{N}{m}$, (2) $u = upper \times \frac{N}{m}$, where N is the data set size and m is the size of the sample data set used in the feature extraction. Also, the range can be adjusted manually to obtain a good clustering result. Once we have identified an interesting cluster, the sub-clusters in its descendent nodes will not be scanned. A **representative feature** is defined as a feature that has enough support in the cluster. That is, the document frequency of the representative features must be greater than some user-defined threshold. We call this threshold the **representative threshold**. These features will then be used to represent the cluster.

5 Experimental Results

In this section, we list some experimental results for the proposed feature extracting method and the Web document clustering algorithm. We have applied the algorithm on both synthetic data and real data for studying the performance and the relationship among the parameters of the proposed algorithms. We find that the behaviour of the algorihtms are similar for the real data and synthetic data. For interest of space, we shall report on either data set in each performance analysis. All experiments are run on the *Sun Ultra 5/270* with 512 MB of RAM. The programs are compiled using *GNU* C++ and executed on the *Solaris 2.6* operating system.

The synthetic datasets are generated by a generator that we have developed. The data generation is controlled by four parameters: (1) the number of document feature vectors, (2) the number of features, (3) the number of clusters, and (4) the **similarity level** L which is ranged from 0 to 1. Given x feature vectors and y clusters, for each cluster C, we set x/y features F randomly to represent the cluster. For each document generated for C, the features in F will have a chance of L being set to 1 and $1-L$ being set to 0. The features not in F will have a chance of $1-L$ being set to 1. Hence the higher the similarity level, the more similar the documents will be within the same cluster.

We have collected two sets of Web pages from the *Yahoo!* search engine. One contains ten non-correlated topics while another contains ten correlated topics. The Web pages indexed by *Yahoo!* search engine have already been classified.

To construct the non-correlated document set, we have selected ten subtopics, namely, author, credit bank, internet game, wine, film, disease organization, job opportunity, soccer, astronomy organization, psychology department. For each topics, we collected 2000 Web pages. So, there are totally 20000 Web pages in this dataset.

For the correlated topic Web document set, we have selected ten subtopics from the following location:
`http://dir.yahoo.com/Computers_and_Internet/Programming_Languages/`
from *Yahoo!* search engine, namely, COBOL, Fortran, Java, JavaSript, Lisp, Pascal, Perl, Python, Smalltalk and Visual Basic. For each topics, we collected 2000 Web pages. So, there are also totally 20000 Web pages in this dataset.

We use the two real data sets of 20000 web documents each for both the correlated topics and non-correlated topics.

	non-correlated topics	correlated topics
No. of clusters found	10	10
Cluster size range	1000-3000	1000-3000
Representative threshold	0.45	0.4
Clustering time	94.21 seconds	72.27 seconds
F-Measure Value	0.6753	0.6453

Table 1. Real data clustering results

The representative thresholds for the DC-tree method in the two cases are shown in Table 1, and are chosen by trial and error.

5.1 Precision

For each cluster found, we calculate the document support of each known topic, which is the number of documents containing the same topic. Then we choose the topic with the highest document support as the topic of the cluster. Tables 2 (a) and (b) show the precision values of the clusters found by these two algorithms. They show that the DC-tree always gives a better clustering result than B^+-tree in terms of the precision of the resulting clusters.

Table 3 also lists the representative features of each cluster. Since the classification is automatic, we do not have a simple title for each topic of the clusters found, but we can guess the topic of the clusters by their representative features. The representative features (words) are used to indicate to the users what each cluster is about.

Cluster Topic	Precision	
	DC-tree	B^+-tree
author	0.61	0.59
internet game	0.73	0.55
wine	0.63	0.65
credit bank	0.66	0.57
soccer	0.73	0.47
astronomy organization	0.88	0.77
psychology department	0.85	0.63
job opportunity	0.91	0.72
film	0.72	0.73
disease organization	0.64	0.65
Average	**0.736**	**0.633**

(a)

Cluster Topic	Precision	
	DC-tree	B^+-tree
Cobol	0.64	0.54
Fortran	0.72	0.72
Java	0.73	0.62
JavaScript	0.64	0.67
Lisp	0.73	0.63
Pascal	0.81	0.67
Perl	0.78	0.72
Python	0.82	0.55
Smalltalk	0.61	0.57
VisualBasic	0.77	0.65
Average	**0.725**	**0.634**

(b)

Table 2. Precision of the clusters (a) non-correlated topics (b) correlated topic

Cluster Topic	No. of Documents	Features
author	2157	book, story, year
internet game	1988	best, free, game, review
wine	2461	wine
credit bank	2302	credit, service, union
soccer	1312	report, soccer, team
astronomy organization	1712	astronomy,union
psychology department	1892	psychology, university
job opportunity	1939	career, experience, job, location
film	1664	film, star
disease organization	2245	disease, health, medical, organization

Table 3. The clusters found for non-correlated data

5.2 Incremental updates on Real Data Set

In order to study the impact of the incremental updating the clustering result of the non-correlated topic document set, we insert 5000 additional documents into the existing DC-tree and observe the change of clustering quality. 433.22 seconds are used to convert the 5000 documents into feature vectors. Table 4 shows the changes of clustering quality for each 1000 additional documents added to the existing clustering result. From the table, we notice that the F-Measure value decreases from 0.6753 to 0.6010. The clustering quality only decreases about 11% in terms of F-Measure value when 25% more documents are added to the existing document set. Note also that each update in the document set will only involve a very small amount of computation in the clustering. Therefore, our document clustering algorithm is suitable for incremental updating in Web pages.

No. of documents	F-Measure value
21000	0.6555
22000	0.6475
23000	0.6249
24000	0.6062
25000	0.6010

Table 4. Incremental Update Results

6 Conclusion

In this paper, we introduce a tree structure called DC-tree for the incremental and hierarchical Web document clustering. We propose a feature extracting method for document clustering in the Web domain. This method takes into account the special characteristics of Web pages, and the special requirements of the problem domain. We show by experiment that the features of the DC-tree give good results in terms of accuracy and efficiency.

References

1. A. Guttman. R-tree: A dynamic index structure for spatial searching. In Proceedings of ACM SIGMOD, pages 47-56, 1984.
2. Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 16-22, San Diego, CA, USA, August 1999.
3. C. J. van Rijsbergen. Information Retrieval. Butterworth & Co (Publishers) Ltd, 1979.
4. Wai-chiu Wong and Ada Fu, Incremental Document Clustering for Web Page Classification, IEEE 2000 Int. Conf. on Info. Society in the 21st century: emerging technologies and new challenges (IS2000), Nov 5-8, 2000, Japan
5. Clement T. Yu and Weiyi Meng. Principles of Database Query Processing for Advanced Applications, chapter 10, pages 359-385. Morgan Kaufmann Publisher, Inc., 1998.
6. Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In Proceedings of the ACM SIGMOD Conference on Management of Data, pages 103-114, Montreal, Canada, June 1996.