Incremental Document Clustering for Web Page Classification

Wai-chiu Wong and Ada Wai-chee Fu
Department of Computer Science and Engineering,
Chinese University of Hong Kong, Hong Kong.
E-mail: adafu@cse.cuhk.edu.hk

Abstract—**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.**

Keywords—**Incremental update, Tree, Document, Clustering, Web, Classification**

I. Introduction

THE popularity of the Internet has caused a continuous massive increase in the amount of Web pages (or Web documents) [1]. The information explosion has led to a growing challenge for information retrieval systems, such as the Internet search engine, to efficiently and effectively retrieve this information. Document classification becomes an important process for helping the information retrieval systems organize this vast amount of data. For instance, most of the Internet search engines will divide the indexed Web documents into a number of classes for the users to limit the search scope. Moreover, document classification can make the retrieved results easier to browse. Most search engines return a large amount of results when a simple query (one or two words) is submitted. It can be difficult for the users to find the documents they want. If the results returned by the Internet search engine have been classified into several groups with some descriptions, the users can choose the interesting group(s) to continue browsing.

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 [1], [3], [4], [5]. 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 predetermined 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.

Many clustering algorithms have been introduced for high dimensional data mining, for example, K-Means, CLARANS [6], BIRCH [7], CURE [8], CLIQUE [9] and ENCLUS [10]. However, the existing clustering algorithms in general are not suitable to solve the document clustering problem. Most of them (such as K-Means, CLARANS and BIRCH) require the supply of the number of clusters to work on. Unfortunately, the number of clusters or classes of the document set is usually unknown to user. Also, a major issue that document databases are now facing is the high rate of update. Researchers have found that existing clustering algorithms are not suitable for maintaining clusters in such a dynamic environment, and they have identified the problem of updating clusters without frequently performing complete re-clustering [11], [12]. Here, we propose to make use of a tree structure for incremental document clustering.

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 doc-

[1] We shall use the terms *Web pages* and *Web documents* and *documents* interchangeably.

ument insertion order and is more tolerant towards noise documents.

Feature selection is a major problem related to document clustering. We find that traditional feature extraction methods are not suitable in the Web domain. These feature extraction methods select the n highest-weighted terms as the features. The term weighting scheme is commonly based on the term frequency (TF) or the term frequency combined with the inverse document frequency (TF-IDF). However, we discover from experiments (see Section II-A) that a Web page usually contains a small number of words and most words appear only one to two times. Hence the term frequency of a word cannot indicate the actual importance of this word. This method also ignores the *coverage* [2] of the features extracted. The coverage is important because we do not want many tiny clusters to appear. We propose a feature extraction algorithm which is more suitable for the problem domain.

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.

This paper is organized as follows. Section II gives the background information and introduces our feature extraction method. In Section III, the DC-tree structure and corresponding insertion algorithm will be described in detail. Section IV reports on the experimental evaluation of our proposed feature extracting method and the Web document clustering algorithm.

II. 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. We describe three alternatives for measuring the similarity between two document feature vectors. Finally, the clustering quality evaluation method will be presented.

A. Statistical Analysis in the Web Domain

In order to study whether the traditional term weighting is suitable for the Web document clustering, we have carried out an experiment to analyze the number of distinct words and the average word frequency of the Web pages. We have used the stratified random sampling technique for collecting the document set in this experiment. Ten topics (Arts & Humanities, Business & Economy, Computers & Internet, Entertainment, Government, Health, Recreation & Sports, Science, Social Science, and Society & Culture) have been chosen from the *Yahoo!* search engine. Then we randomly collect 2000 documents for each topic. While collecting the Web documents, the number of *valid words* [3] of the pages is counted. We will discard any document that contains zero valid word. Totally 20000 Web pages have been collected from the *Yahoo!* search engine. Some statistics about the words of Web documents are shown in Table I. S.D. is the standard deviation.

	No. of distinct words	**Average** word freq.
Mean	174.75	1.68
Median	116.00	1.55
S.D.	225.82	0.71
Maximum	4351	17.59
Minimum	1	1.0

TABLE I
Statistics about the words of Web documents

We made a study on the number of distinct words in the Web documents. We notice that most of the Web documents contain less than 500 words. According to statistics, there are 17027 (85.14%) documents that contain less than or equal to 300 distinct words, 18930 (94.65%) documents that contain less than or equal to 500 distinct words, and 19712 (98.56%) documents that contain less than or equal to 1000 distinct words

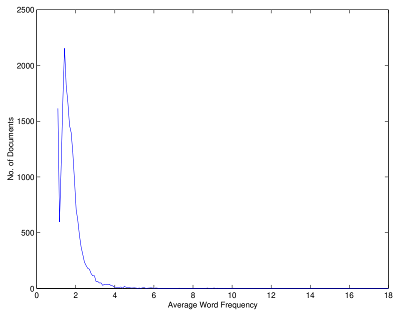

Fig. 1. Document distribution along the average word frequency

According to Figure 1 and Table I, the average word frequency (or term frequency) of almost all documents is less than 2.0 which means that most of words in a Web document will rarely appear more than 2 times. Therefore, the traditional term frequency weighting scheme which was designed for longer documents may not be useful in Web Domain.

B. Document Feature Extraction

Luhn [17] proposed that the frequency of word occurrences in an article furnishes a useful measurement of word significance. Let us order the words by their frequency of occurrences, resulting in the *rank order*. According to *Zipf's Law* [18] (see also [19]), the product of the frequency

[2] The *coverage* is the percentage of documents that contain the extracted features.

[3] Here, the *valid word* is define as the remaining words of the page after stop-word removal and word stemming process.

of use of words and the rank order is approximately constant.

Traditionally, a document D is represented by a feature vector of the form $(d_1, \ldots, d_n)$ where d_i is the numeric weight for the i-th feature and n is the total number of features. Each feature corresponds to a word appearing in the training corpus after stemming the words and the removal of the stop-words. The following is a common feature extracting method [20] which is based on the number of occurrences of particular terms in the documents: (1) Eliminate common function words from the document texts by consulting a special dictionary, or stop-list, containing a list of high-frequency function words. (2) Compute the **term frequency** tf_{ij} for all remaining terms T_j in each document D_i, specifying the number of occurrences of T_j in D_i. Another method combines the term frequency with the inverse document frequency (TF-IDF). The **document frequency** df_j is the number of documents in a collection of N documents in which term T_j occurs. A typical inverse document frequency (idf) factor [24] of this type is given by $log(N/df_j)$. The weight of a term T_j in a document D_i is given by $w_{ij} = tf_{ij} \times log(\frac{N}{df_j})$. Then n terms with the highest weights in all the documents are chosen as the n features.

However, from experiments we find that the above feature extracting methods are not suitable to the Web document clustering algorithm due to the following reasons: (1) As shown in Section II-A, a Web document usually contains a small number of words (about 100-300 words). So, the term frequency of the key word may not be higher than that of the non-key word. (2) Usually, the users do not want too many clusters in the clustering result. However, the TF-IDF method may extract those features with relatively low document frequency because the lower document frequency will give a higher weight of the feature. Such features can lead to a clustering result containing many small clusters. (3) The above methods also ignore the *coverage* of the features extracted. The **coverage** of the features is defined as the percentage of documents containing at least one feature of the features extracted. Since the above methods only choose the best n terms, there is no guarantee that the n terms cover a high percentage of the documents. If the *coverage* is too low, there will be many documents represented by a feature vector with zero weight in all entries.

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 predefined threshold. If so, stop. If no, set $lower = lower - 1$ and $upper = upper + 1$, and then goto Step 5.

Since the document set is usually very large, it is inefficient to carry out the feature extraction process on the entire document set. 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 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 which are not suitable to represent the concept of a class. 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. From the above, 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 Section 5), we find that 0.8 is a good coverage threshold value.

C. 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 II-B, 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.

D. 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 Definition) 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: (DC 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.

We have considered three alternatives for measuring the similarity between two document feature vectors based on the classical vector space model [22]. The first measure is called *cosine correlation* which is commonly used to measure the similarity between the feature vectors of two documents (e.g. [1], [23], [3], [5]). A larger cosine value indicates that they are more similar. The second similarity measuring method uses the *Euclidean distance*, which is the most common similarity measuring method in high dimensional data clustering [8], [6], [7]. The third alternative is the *Manhattan distance*.

We have performed some experiments and found that the Manhattan distance gives similar or better results but uses less computation time compared with the other methods. Hence we have chosen Manhattan distance for our similarity measurement.

E. Evaluation Techniques

In order to evaluate the quality of the clustering result, we adopt the *F-Measure* evaluation technique which was first introduced in [13] (see also [2]). 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 [2], [13] 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 [23] 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.

III. 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 [14] 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 2). 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 2 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:

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

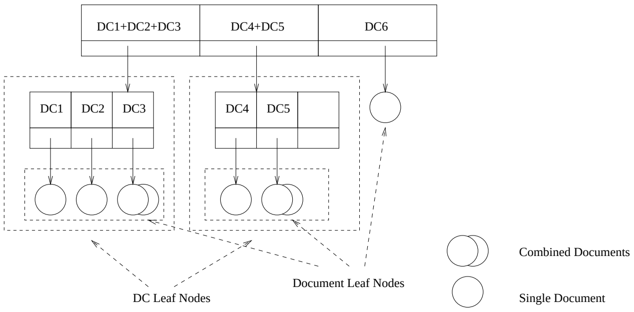

Fig. 2. A DC-tree Example

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 increase 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.

A. Insertion

We now present the algorithm for inserting a document object into a DC-tree. Here the document object can be either a single document, or a cluster of documents which is represented by a DC entry (E). If the document object is a single document, it will be first encapsulated into a DC entry (E) (i.e E contains only one document). The insertion algorithm proceeds in the steps as shown below:

1. **Identifying the appropriate leaf node:** Starting from the root, E recursively descends the DC-tree by choosing the *closest* child node with the similarity value higher than S_1. If such child node does not exist, E is inserted as a new document leaf node to an empty entry of the node. If there is no such empty entry, node *splitting* is required (see Section III-B).

2. **Modifying the leaf node:** When we reach a DC leaf node, we find the *closest* leaf entry to E, say L_i, and then tests whether it can be combined with E without violating the similarity threshold requirement of S_2. If so, the DC entry for L_i is updated to reflect this combination. Note that the DC entry of the new cluster can be computed from the DC entries for L_i and E according to Lemma 1. Otherwise, E is added to the leaf node. If there is space in the leaf node for this new entry to fit in, we are done, otherwise we must *split* the leaf node. Section III-B will discuss the node splitting algorithm in detail.

3. **Modifying the path from the leaf node to the root:** After E is inserted into a leaf node, we must update each non-leaf entry on the path from the root leading to the leaf node. In the absence of a split, this simply involves adding DC entries to reflect the addition of E. A leaf node split requires us to insert a new non-leaf entry into the parent node, which points to the newly created leaf node. If the parent node has space for this entry to fit in, then at all higher levels, we only need to update the DC entries to reflect the addition of the E. In general, however, we may have to *split* the parent node as well, and so on up to the root. If the root is split, the height of the tree will be increased by one and a new root will be created.

B. Node Splitting

In order to add a new entry to a full node containing B entries, it is necessary to divide the collection of the $B+1$ entries between two nodes. The division should be done in a way such that the similarity between the two new nodes will be minimized and the similarity among the documents within the same node will be maximized. We now turn to algorithms for partitioning a set of $B+1$ entries into two groups, one for each new node. The most straightforward way to find the optimal node split is to generate all possible groupings and choose the best. However, the number of possibilities is very large (i.e. approximately 2^{B-1}). Below is the node splitting algorithm used in our DC-tree insertion algorithm. This node splitting algorithm is similar to a method considered in R-tree [25].

1. **Pick a seed for each group:** For each pair of entries E_1 and E_2, compute the similarity between them. Choose the pair with the lowest similarity value as the first elements of the groups. Resolve ties by choosing the pair with the largest number of documents.

2. **Check if done:** If all entries have been assigned, stop. If one group has so few entries that all the rest must be assigned to it in order for it to have the minimum number M, assign them and stop.

3. **Select entry to assign:** For each entry E not yet in a group, calculate the similarity between E and the seed entry of each group. Assign the entry with the highest similarity value into the corresponding group. Goto step 2.

This splitting algorithm attempts to find a good node split in quadratic time in terms of B, but it is not guaranteed to find the optimal solution.

C. Deletion and Node Merging

The data deletion algorithm is similar to that of the B^+-tree. If the number of remaining entries is larger than or equal to the minimum entries number parameter M after removing an entry, the deletion operation is done. Otherwise, node-merge operation should be carried out. That is, the underflow node will be merged with its sibling. Further node-merge operation is needed when the entry underflow phenomenon occurs in the parent node. This operation can be propagated to the root node where the height of tree may be reduced if necessary.

D. 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 (see Section II-B) 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.

IV. Experimental Results

In this section, we list some experimental results for the proposed feature extracting method and the Web document clustering algorithm. A more complete set of results can be found in [26]. 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. 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.

A. Synthetic Data Generation

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.

B. Yahoo! Web Page Collections

We have collected two sets of Web pages from the *Yahoo!* search engine [15]. 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 sub-topics, 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 sub-topics 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.

C. Comparing DC-tree and B^+-tree

In this experiment, we want to make a comparison between the effectiveness of the document clustering in Web domain using the DC-tree and the B^+-tree. We use the two real data sets of 20000 web documents each for both the correlated topics and non-correlated topics. We use two of the feature sets found in our experiment for converting each document to a feature vector in this experiment. This set contains 119 features. The DC-tree parameters are the same as the previous experiment. We employ the binary weighting schemes to construct the document vectors and apply the document clustering algorithms to cluster these document vectors. We compare their effectiveness in terms of the F-Measure value of the clustering result and also on a precision value.

	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 sec	72.27 sec
F-Measure Value	0.6753	0.6453

TABLE II
Real data clustering results

The representative thresholds for the DC-tree method in the two cases are shown in Table II, and are chosen by trial and error. The table also shows some experimental measurements. The experimental results in terms of F-measure on the non-correlated topic document set and the correlated topic document set are shown in Figures 3 and IV-C, respectively. The figures show that DC-tree always gives a better clustering result than the B^+-tree does for both real data sets in terms of F-measure.

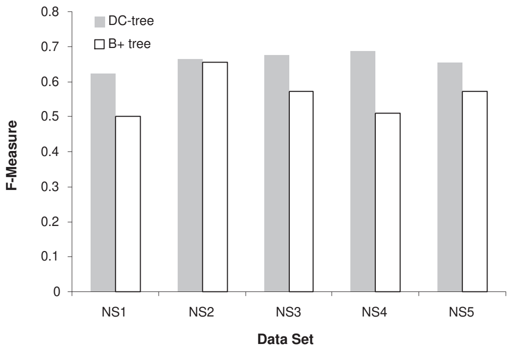

Fig. 3. F-Measure for non-correlated topic

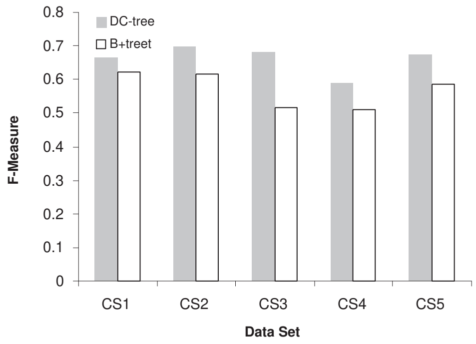

Fig. 4. F-Measure for correlated topic document set

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**

TABLE III
PRECISION OF THE CLUSTERS FOR THE NON-CORRELATED TOPIC DOCUMENT SET

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**

TABLE IV
PRECISION OF THE CLUSTERS FOR CORRELATED TOPIC

C.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 IV and III 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. The results are also quite high compared to results reported in [16] in terms of the average precision. In [16], the precision of the best precision results shown are about 0.4. Though we cannot compare directly with [16] since the data sets used are different, the results are very encouraging.

Table V and Table VI 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.

D. 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 in the last experiment, 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 VII 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.

Cluster Topic	Features
author	book, story, year
internet game	best, free, game, review
wine	wine
credit bank	credit, service, union
soccer	report, soccer, team
astronomy organization	astronomy,union
psychology department	psychology, university
job opportunity	career, experience, job, location
film	film, star
disease organization	disease, health, medical, organization

TABLE V

THE CLUSTERS FOUND FOR NON-CORRELATED DATA

Cluster Topic	Features
Cobol	cobol, coughlan, procedure
Fortran	eacute, fortran
Java	applet, class, event
JavaScript	active, document, javascript
Lisp	lisp, object
Pascal	define, interface, pascal, xplain
Perl	cpan, module, perl, port
Python	kendall, python, thread
Smalltalk	browse, develop, message, smalltalk
VisualBasic	basic, click, item, visual

TABLE VI

CLASSIFICATION RESULT FOR THE CORRELATED TOPIC WEB PAGE COLLECTION

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

TABLE VII

INCREMENTAL UPDATE RESULTS

REFERENCES

[1] Wai Lam and Chao Yang Ho, Using A Generalized Instance Set for Automatic Text Categorization, Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, 81-89, Melbourne, Australia, August, 1998.

[2] David D. Lewis and William A. Gale, A Sequential Algorithm for Training Text Classifiers, Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, 3-12, Dublin, Ireland, July, 1994.

[3] David D. Lewis and Robert E. Schapire and James P. Callan and Ron Papka, Training Algorithms for Linear Text Classifiers, Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval, 298-306, Zurich, Switzerland, August, 1996.

[4] Seán Slattery and Mark Craven, Combining Statistical and Relation Methods for Learning in Hypertext Domains, Proceedings of the 8th International Conference on Inductive Logic Programming, Madison, Wisconsin, USA, July, 1998.

[5] Yiming Yang, Expert Network: Effective and Efficient Learning from Human Decisions in Text Categorization and Retrieval, Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, 13-22, Dublin, Ireland, July, 1994

[6] Raymond T. Ng and Jiawei Han, Efficient and Effective Clustering Methods for Spatial Data Mining, Proceedings of the 20th VLDB Conference, 144-155, Santiago de Chile, Chile, September, 1994.

[7] Tian Zhang and Raghu Ramakrishnan and Miron Livny, BIRCH: An Efficient Data Clustering Method for Very Large Databases, Proceedings of the ACM SIGMOD Conference on Management of Data, 103-114, Montreal, Canada, June, 1996.

[8] Guha and Rajeev Rastogi and Kyuseok Shim, CURE: An Efficient Clustering Algorithm for Large Databases, Proceedings of the ACM SIGMOD Conference on Management of Data, 73-84, Seattle, Washington, USA, June, 1998.

[9] R. Agrawal and Johannes Gehrke and Dimitrios Gunopulos and Prabhakar Ra ghavan, Automatic Subspace Clustering of High Dimensional Data for Date Mining A pplications, Proceedings of the ACM SIGMOD Conference on Management of Data, 94-105, Seattle, Washington, USA, June, 1998.

[10] Chun-hung Cheng and Ada Wai-chee Fu and Yi Zhang, Entropy-based Subspace Clustering for Mining Numerical Data, Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 84-93, San Diego, CA, USA, August, 1999.

[11] F. Can and N. D. Drochak II, Incremental Clustering for Dynamic Document Databases, Proceedings of the 1990 Symposium on Applied Computing, 61-67, 1990.

[12] Moses Charikar and Chandra Chekuri and Tomas Feder and Rajeev Motwani, Incremental Clustering and Dynamic Information Retrieval, Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory o f Computing, 626-635, El Paso, Texas, USA, May, 1997.

[13] van Rijsbergen and C. J., Information Retrieval, Butterworth & Co (Publishers) Ltd, 1979.

[14] Clement T. Yu and Weiyi Meng, Principles of Database Query Processing for Advanced Applications, Morgan Kaufmann Publisher, Inc., 359-385, 1998.

[15] Yahoo Search Engine, http://www.yahoo.com.

[16] Oren Zamir and Oren Etzioni, Web Document Clustering: A Feasibility Demonstration, Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, 46-54, Melbourne, Australia, August, 1998.

[17] H. P. Luhn, The automatic creation of literature abstracts, IBM Journal of Research and Development, 2, 159-165, 1958.

[18] H. P. Zipf, Addison-Wesley, Human Behavior and the Principle of Least Effort, Cambridge, Massachusetts, 1949.

[19] R. A. Fairthorne, Empirical hyperbolic distributions (Bradford-Zipf-Mandelbrot) for biliometric description and prediction, Journal of Documentation, 25, 319-343, 1969.

[20] Gerard Salton, 9, Addison-Wesley Publishing Company, Automatic Text Processing. The Transformation, Analysis, and Retrieval of Information by Computer, 275-312, 1989.

[21] Gerard Salton and M. J. McGill, McGraw-Hill, Introduction to Modern Information Retrieval, New York, 1983.

[22] , Gerard Salton and M. J. McGill, McGraw-Hill, Introduction to Modern Information Retrieval, New York, 120-123, 1983.

[23] Bjornar Larsen and Chinatsu Aone, Fast and Effective Text Mining Using Linear-time Document Clustering, Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 16-22, San Diego, CA, USA, August, 1999.

[24] , Gerard Salton and M. J. McGill, McGraw-Hill, Introduction to Modern Information Retrieval, New York, 63, 1983.

[25] A. Guttman, Proceedings of ACM SIGMOD, R-tree: A Dynamic Index Structure for Spatial Searching, 47-56, 1984.

[26] Wai-chiu Wong, Ada Fu Incremental Document Clustering for Web Page Classification, manuscript, full version, 2000.