# Web Mining for Unsupervised Classification

戴瑋彥　Wei-Yen Day
國立臺灣大學資訊工程學系
Department of Computer Science and Information Engineering
National Taiwan University
r97067@csie.ntu.edu.tw

紀均易　Chun-Yi Chi
國立臺灣大學資訊工程學系
Department of Computer Science and Information Engineering
National Taiwan University
ono.ccy@gmail.com

陳瑞呈　Ruey-Cheng Chen
國立臺灣大學資訊工程學系
Department of Computer Science and Information Engineering
National Taiwan University
cobain@turing.csie.ntu.edu.tw

鄭卜壬　Pu-Jen Cheng
國立臺灣大學資訊工程學系
Department of Computer Science and Information Engineering
National Taiwan University
pjcheng@csie.ntu.edu.tw

劉培森　Pei-Sen Liu
資訊工業策進會
Institute for Information Industry
psliu@iii.org.tw

## Abstract

Data acquisition is a major concern in text classification. The excessive human efforts required by conventional methods to build up quality training collection might not always be available to research workers. In this paper, we look into possibilities to automatically collect training data by sampling the Web with a set of given class names. The basic idea is to populate appropriate keywords and submit them as queries to search engines for acquiring training data. Two methods are presented in this study: One method is based on sampling the common concepts among the classes, and the other based on sampling the discriminative concepts for each class. A series of experiments were carried out independently on two different datasets, and the result shows that the proposed methods significantly improve classifier performance even without using manually labeled training data. Our strategy for

retrieving Web samples, we find that, is substantially helpful in conventional document classification in terms of accuracy and efficiency.

Keywords: Unsupervised classification, text classification, Web mining

## 1. Introduction

Document classification has been extensively studied in the fields of data mining and machine learning. Conventionally, document classification is a supervised learning task [1, 2] in which adequately labeled documents should be given so that various classification models, i.e., classifiers, can be learned accordingly. However, such requirement for supervised text classification has its limitations in practice. First, the cost to manually label sufficient amount of training documents can be high. Secondly, the quality of labor works is suspicious, especially when one is unfamiliar with the topics of given classes. Thirdly, in certain applications, such as email spam filtering, prototypes for documents considered as spam might change over time, and the need to access the dynamic training corpora specifically-tailored for this kind of application emerges. Automatic methods for data acquisition, therefore, can be very important in real-world classification work and require further exploration.

Previous works on automatic acquisition of training sets can be divided in two types. One of which focused on augmenting a small number of labeled training documents with a large pool of unlabeled documents. The key idea from these works is to train an initial classifier to label the unlabeled documents and uses the newly-labeled data to retrain the classifier iteratively. Although classifying unlabeled data is efficient, human effort is still involved in the beginning of the training process.

The other type of work focused on collecting training data from the Web. As more data is being put on the Web every day, there is a great potential to exploit the Web and devise algorithms that automatically fetch effective training data for diverse topics. A major challenge for Web-based methods is the way to locate quality training data by sending effective queries, e.g., class names, to search engines. This type of works can be found in [3, 4, 5, 6], which present an approach that assumes the search results initially returned from a class name are relevant to the class. Then the search results are treated as auto-labeled and additional associated terms with the class names are extracted from the labeled data. By sending the class names together with the associated terms, appropriate training documents can be retrieved automatically. Although generating queries is more convenient than manually collecting training data, the quality of the initial search results may not always be good especially when the given classes have multiple concepts. For example, the concepts of class “Apple” include company and fruit. Such a problem can be observed widely in various applications.

The goal of this paper is, given a set of concept classes, to automatically acquire training corpus based merely on the names of the given classes. Similar to our previous attempts, we

employ a technique to produce keywords by expanding the concepts encompassed in the class names, query the search engines, and use the returned snippets as training instances in the subsequent classification tasks. Two issues may arise with this technique. First, the given class names are usually very short and ambiguous, making search results less relevant to the classes. Secondly, the expanded keywords generated from different classes may be very close to each other so that the corresponding search-result snippets have little discrimination power to distinguish one class from the others.

We present two concept expansion methods to deal with these problems, respectively. The first method, expansion by common concepts, aims at alleviating the problem of ambiguous class names. The method utilizes the relations among the classes to discover their common concepts. For example, "company" could be one of the common concepts of classes "Apple" and "Microsoft". Combined with the common concepts, relevant training documents to the given classes can be retrieved. The second method, expansion by discriminative concepts, aims at finding discriminative concepts among the given classes. For example, "iPod" could be one of the unique concepts of class "Apple". Combined with the discriminative concepts, effective training documents that distinguish one class from another can be retrieved.

Our methods are tested under two different experimental setups, the CS papers and Web pages classification tasks. The proposed methods are effective in retrieving quality training data by querying search engines. Moreover, the result shows that the obtained Web training data and manually labeled training data are complementary. Our methods can significantly improve classification accuracy when only a few manually labeled training data is available. Contribution of our work can be addressed as follows. We propose an automatic way to sample the Web and collect the training data with good quality. Apart from the previous work, our methods are fully automatic, reliable, and robust, and achieve an 81% accuracy in text classification tasks. With a little help from a small number of labeled data added into the scene, the classification accuracy can be as high up to 90%. Several experiment results are also revealed to help investigation and realization of automatic Web sampling methods, in which the difficulties encountered are presented in detail.

The sections are organized as follows. In Sections 2 and 3, we present our basic idea and the two methodologies, respectively. The experiments are introduced in Section 4. In Section 5, we discuss the related work of this paper. Finally, in Section 6, we give out discussions and conclusions.

## 2. The Basic Idea

Suppose that we are given a set of classes $C=(c_1, c_2, \ldots, c_n)$, where $c_i$ is the name of the i-th class. We plan to generate keywords based on classes $C$, form a few queries and send them off to search engines so as to collect training instances. Our methods presented in this paper are independent of classification models; that is, any model can be incorporated with our methods.

To carefully examine the possibility of querying search engines for acquiring training data, we did an evaluation with different search engines, search-result types (snippet or document), and the number of search results. 5 CS-related classes were taken into account, including "Architecture", "IR", "Network", "Programming", and "Theory". Each class name $c_i$ was sent to 3 search engines, including Google[1], Yahoo![2], and Live Search[3]. Top 100 snippets were extracted as training data. We also gathered the research papers from the corresponding conferences to the 5 classes as the testing documents. Table 1 shows the performance of different search engines. Querying by the class names can achieve classification accuracy at a range from 0.35 to 0.56. More specifically, the three search engines perform well in "Programming" and "Theory" but poorly in the others on average. This arises from the fact that irrelevant documents may be located for those classes with ambiguous names. The way to query by the class names is not reliable due to the ambiguity of the class names (the first challenge). For example, the word "architecture" is widely used in CS, art and construction. From the results, we select Google as our backend search engine in this paper.

We further explore if the classification performance can be improved by downloading Web pages for training. The result is shown in Table 2. It reveals that Web pages might introduce more noises than snippets do, while the snippets summarize Web pages and capture the concepts of classes C by their context. Moreover, to download Web pages is time-consuming. Our methods, therefore, only retrieve snippets as the training source.

Table 1. Accuracy of different search engines for classification of CS papers.

| Engine | **Architecture** | **IR** | **Network** | **Programming** | **Theory** | **Avg.** |
|---|---|---|---|---|---|---|
| Google | 0.075 | 0.382 | 0.899 | 0.723 | 0.762 | 0.568 |
| Yahoo! | 0.112 | 0.022 | 0.094 | 0.863 | 0.665 | 0.351 |
| Live Search | 0.269 | 0.006 | 0.083 | 0.784 | 0.815 | 0.391 |

Intuitively, collecting more snippets or documents might enhance the performance. Table 3 shows the results of changing training data sizes from 100 to 900. It could be found that classification accuracy does not increase obviously when the numbers of snippets and documents reach 200 and 300, respectively. This is because much relevant information can be retrieved in top ranked search results returned by the search engine. Noises are unavoidably included from longer lists. Hence, simply fetching a large amount of snippets or documents from a single search result cannot achieve satisfactory performance. Even if we expand the queries, i.e., the class names, using pseudo-relevance feedback (PRF) [7, 8, 9], the improvement is still minor since the generated expanded keywords cannot effectively discriminate different classes (our second challenge). The performance comparison between our methods and PRF will be given in Section 4.2.

[1] The Google search engine: http://www.google.com/search
[2] The Yahoo! search engine: http://search.yahoo.com/search
[3] The MSN Live search engine: http://search.live.com/

To collect good training corpora and help classifiers learn more quickly (querying search engines is costly), two methods are proposed in this paper. The first method, expansion by common concepts, aims at alleviating the ambiguity problem. Generally, a short class name easily conveys multiple meanings. For example, class "Apple" may be a fruit or company name. We find that class $c_i$ is context-aware if its context $C$ - $\{c_i\}$ provides relevant information to $c_i$. For example, if "Apple" and "Microsoft" are put together, "Apple" would be a company. If we are given "apple" and "banana", "apple" could refer to a fruit. Our first method, which will be described in Section 3.1, is trying to discover such common concepts among classes *C*, i.e., "company" and "fruit", from the Web, and use them as constraints to expand our original queries *C*.

Table 2. Accuracy of different training types in CS papers classification.

| Source | **Architecture** | **IR** | **Network** | **Programming** | **Theory** | **Avg.** |
|---|---|---|---|---|---|---|
| Snippet | 0.075 | 0.382 | 0.899 | 0.723 | 0.762 | 0.568 |
| Documents | 0.272 | 0.049 | 0.689 | 0.505 | 0.783 | 0.459 |

Table 3. Average accuracy of different training sizes in CS papers classification.

| # of docs | **100** | **200** | **300** | **400** | **500** | **600** | **700** | **800** | **900** |
|---|---|---|---|---|---|---|---|---|---|
| Snippets | 0.568 | 0.601 | 0.603 | 0.601 | 0.608 | 0.603 | 0.604 | 0.604 | 0.604 |
| Documents | 0.460 | 0.463 | 0.507 | 0.500 | 0.503 | 0.504 | 0.507 | 0.507 | 0.508 |

Common concepts can help us collect more relevant documents to each class but cannot discriminate one class from the others. The latter becomes important because classification is inherently to distinguish different classes. Our second method is focused on the finding of discriminative concepts among classes *C*. Consider previous example. "PowerPoint" and "iPod" are possible discriminative concepts because "PowerPoint" is only relevant to "Microsoft" while "iPod" is only about "Apple". Different from PRF, our second method, expansion by discriminative concepts, which will be described in Section 3.2, aims at acquiring Web training data not only relevant to each class but also effectively distinguishing one class from another.

## 3. The Proposed Methods

In this section, we will describe the two training data acquiring methods, sampling the Web by common concepts expansion and discriminative concepts expansion, respectively.

### 3.1 Expansion by Common Concepts

The goal of our first method is to collect training data via sampling the Web by discovering the common concepts between given classes *C*. Expanding class names *C* by their common concepts is helpful in obtaining more suitable training data from search engines. An intuitive way to discover the concepts is to find common concepts from well-known topic hierarchies on the Web such as Open Directory Project[4] (DMOZ), which is one of the largest and comprehensive human-edited directories. To obtain common concepts, we first search

[4] DMOZ Open Directory Project: http://www.dmoz.org/

DMOZ by each class name $c_i$ and get a set of the nodes relevant to $c_i$ in the DMOZ directory. Suppose the set of the nodes is $N(c_i)$. The least common ancestors (LCA) of all of the nodes $N(c_i)$ $(i=1...n)$ are viewed as the common concepts of $C$. The LCAs are the shared ancestors of $N(c_i)$ $(i=1...n)$ located farthest from the roots of the DMOZ directory. For example, by searching class "Architecture", we get the path "Arts: Architecture" and "Computers: Emulators: Intel x86 Architecture". When search the class "Programming", we find it has the same common concept of "Computers" (from "Computers: Programming") with "Architecture". Thus the concept "Computers" would be the common concepts between the two classes "Architecture" and "Programming".

Although Open Directory Project covers diverse topics and is very precise, sometimes we might get few or even no common concepts among $C$. The problem is serious for those classes not so popular such as names of person or organizations. For example, if we query the class "Cornell", we only get 5 paths for the class, which contains few candidates of concepts to select and expand. To deal with this problem, we extract terms co-occurring with each class $c_i$ in Web pages, cluster the terms, and treat the representative term for each class as one of the common concepts. More specifically, all of the classes $\{c_1, c_2, \ldots, c_n\} \in C$ are combined into one query "$c_1 + c_2 + \ldots + c_n$", and then submitted to a search engine. After stemming and removing stopwords, we extract 20 high-frequency terms as candidates for common concepts from top 100 snippets returned from the search engine. To group these candidates, we send them separately to the search engine and generate corresponding feature vectors based on their top 100 snippets. Uni- and bi-grams are adopted as feature terms and TF-IDF is used to calculate feature weights. Next, a graph $G = (V, E)$ is constructed, where $v \in V$ represents one candidate term, and $e \in E$ is the cosine similarity between two feature vectors. Finally, we perform the star clustering algorithm [10] to choose the star centers, which are the common concepts among $C$.

In this paper, we adopt both of LCAs from DMOZ and co-occurring terms from Web pages as our common concepts among $C$. After common concepts generated, we can either use it to sample the Web and acquire good training data, or utilize them while discriminative concepts are generating.

### 3.2 Expansion by Discriminative Concepts

A discriminative concept is a concept that can help distinguish one certain concept class of interest (say, $c$) from all the classes ($c' \neq c$). Such a concept contributes more relevance to one specific class than to the others. Unlike common concepts, which are shared by all the concept classes in $C$, any discriminative concept has a specific concept class to contribute relevance to, called the *host*. Let $f_c$ be the feature vector of concept class $c$. Let the similarity between any two concepts $x$ and $y$ be denoted as $\sigma_{x,y} = cos(f_x, f_y)$. An ideal discriminative concept $k$ for concept $c$ must satisfy all the following constraints:

1. Concept $k$ should exhibit high similarity to its host $c$.

2. The similarity between *k* and *c* should be significantly greater than that between *k* and any one of the other concept classes.

These constraints loosely define the criteria that we can use to discover discriminative concepts, and they also rule out the possibility that a concept *k* has two or more hosts. Based on the second constraint, a plausible decision criteria for discriminative concept is given below. Let $\kappa_c$ denote the set of all discriminative concepts hosted by concept class *c*. We have:

$$k \in \kappa \Leftrightarrow \frac{\sigma_{c,k}}{\sigma_{c',k}} > \theta \; for \; all \; c' \neq c$$

In the criteria, the right-hand side equation needs to be satisfied for all other concept classes *c'*; in other words, we have a multiple-constraint-satisfaction problem. For every concept-class pair *(c, c')*, the ratio $\sigma_{c,k}/\sigma_{c',k}$ describes the degree of deviation in similarities exhibited by *k* toward both classes. When the value is greater than 1, we say that *c* is more likely to be the host; when it lies between 0 and 1, we say that *c'* is more likely. The parameter $\theta$ determines the tightness of the decision boundary. Since we expect higher similarity between *k* and *c* than that between *k* and other *c'*s, it would normally be defined as a value greater than 1.

Generally, fixed boundary value is easier to train and suitable for general cases, while we also find that this type of setup can cause problems in extreme cases. Suppose we have two classes $c_1$ and $c_2$ which are relevant topics or extraordinarily similar to each other (in terms of the similarity between their feature vectors). The number of discriminative concepts for either $c_1$ or $c_2$ may drastically decrease because, for most *k*'s, deviation in similarities toward both classes becomes even more subtle and harder to detect by using a simple constant $\theta$. An obvious solution to this problem, we find, is to adopt a function in place of the constant $\theta$, that assigns high threshold value for general cases and low threshold value for the aforementioned extreme cases. In real practice, we use a very simple form of decision criteria to identify discriminative concepts:

$$k \in \kappa \Leftrightarrow \frac{\sigma_{c,k}}{\sigma_{c',k}} > \delta(1 - \sigma_{c,c'}) \text{ for all } c' \neq c$$

where $\delta$ is the *discrimination coefficient*. With the new criteria, a number of 5 to 40 discriminative concepts (for each class) can still be discovered even when any two concept classes exhibit high class-to-class similarity to each other.

The next step is to apply this technique to each concept class in *C* so as to populate new training instances from the Web. Let $SR_c$ be the set of search-result snippets obtained by sending concept *c* as a query to the search engine. Assume that the training set for concept class *c* before and after the expansion is denoted as $D_c$ and $D'_c$, respectively. Given $\kappa_c$, we can form a set of new queries by concatenating *c* with each $k \in \kappa_c$,, send them off to the search engine, and obtain a new set of training instances, i.e., $\{ SR_{c \cup k} \in \kappa_c \mid k \in \kappa_c \}$. In formalism, we have:

$$D'_c = D_c \cup_{k \in \kappa_c} SR_{c \cup k}$$

This procedure can be repeated several times for each class so that the total number of discovered training instances can reach our expectation. However, certain changes on definitions and notations required by the adaptation needs to be clarified in advance. First, we can no longer expect that the feature vector for a concept class *c* remains the same throughout multiple iterations. In each iteration, when new training instances added to the collection, feature vectors for *c* actually changes. Next, the set of discriminative concepts $\kappa_c$ discovered in each iteration would vary, as similarity measure suffers from the change as well. Certain modification in definitions should be taken care of so as to seamlessly fit the aforementioned criteria into the framework, while treating notations for a concept and for a concept class differently might clutter up the framework. For simplicity, no explicit treatment will be done to the equations in the following text. Readers should take caution that, when class-to-concept or class-to-class similarity computation is considered in discussions, the feature vector referred to a concept class is in fact derived from its current training set (rather than $SR_c$). On the other hand, we will use $\kappa_c^{(i)}$ in place of plain $\kappa_c$ in the rest of the work.

Assume that the algorithm repeats *t* times. The initial training data for concept class *c* is denoted as $D_c^{(0)}$, and in each iteration $i \in [1,t]$, a new training set for *c* is produced and represented by $D_c^{(i)}$. Consider a simple framework as follows. For all $c \in C$, we have:

$$\mathrm{D_c}^{(0)} = \mathrm{SR_c}$$
$$\mathrm{D_c}^{(\mathrm{i})} = \mathrm{D_c}^{(\mathrm{i})} \cup_{\mathrm{k} \in \kappa_c{}^{(\mathrm{i})}} \mathrm{SR_{c \cup k}}, \qquad \mathrm{i} > 0$$

where $D_c^{(i)}$ and $\kappa_c^{(i)}$ is the set of training instances and the set of discovered discriminative concepts, respectively, for concept class *c* at iteration *i*. Eventually, the algorithm stops after the *t*-th iteration. The content of $D_c^{(t)}$ will serve as the final training data for all concept class *c*.

Since practically it is infeasible to populate the entire set of $\kappa_c$, several heuristics are involved in creation of the set: 1) We look for terms with high discrimination power (specifically, unigram and bigrams) in the context of $D^{(i-1)}$ using commonly-used information-theoretic measures, such as information gain and inverse document-frequency. These candidates are then examined with the decision criteria and disqualified ones are discarded immediately; 2) candidates that survived the test are ranked accordingly by the score function, which is a simple rewrite of the criteria that indicates the average degree of deviation for the candidate *k*:

$$\frac{1}{|\mathrm{c}| - 1} \sum_{\mathrm{c}' \neq \mathrm{c}} \left[ \frac{\sigma_{\mathrm{c,k}}}{\sigma_{\mathrm{c}',\mathrm{k}}} - \delta(1 - \sigma_{\mathrm{c,c}'}) \right]$$

The summands will not cancel out since the score is calculated for the candidates satisfying the criteria. Generally, testing all the candidate terms may result in an extremely inefficient procedure. In practice, we set up a strict threshold on the information gain and idf in light of reducing the number of candidates. We test only the top *m* concepts selected by the filter in the end. The value *m* is set to be 15 throughout the work.

## 4. Experiments

We evaluate our performance in CS papers and Web pages classification.

### 4.1 Experimental Setup

We conduct experiments on two datasets. First is a set of papers from several CS-related conferences, and there are five classes used for training, including "Architecture", "IR", "Network", "Programming", and "Theory", as shown in Table 4. For the paper dataset, there are about 500 papers in each class. Another dataset is the Web pages collected from four universities, and can be downloaded from the WebKB project[5]. The classes for these universities include "Cornell", "Texas", "Washington", and "Wisconsin". As the original dataset for Web pages is imbalanced, we randomly choose 827 Web pages (i.e. the minimum size of original data among the 4 classes) for classification. Please note that the two datasets are our testing data since the training data are fully collected from the Web. Moreover, the two datasets are quite different in document length and quality. The papers are often longer, well written, and with much useful information about the CS-related classes, while the Web pages might cover more noises and shorter contents.

Our first method of expansion by common concepts is denoted as CM; the second method of expansion by discriminative concepts is denoted as DM; CM+DM is the combination of both methods, where CM is applied first so that search results could be more relevant, and then DM is used to extract discriminative concepts from the relevant search results. With the common concepts extracted from CM, we can use these concepts plus the class name as a whole query and perform DM to iteratively collect the discriminative concepts.

Table 4. The information about the dataset of CS papers for classification.

| **Class** | **# papers** | **From Conferences** |
|---|---|---|
| Architecture | 490 | SIGARCH(04-08), DAC(00-07) |
| IR | 484 | SIGIR(02-07), CIKM(02-07) |
| Network | 446 | SIGCOMM(02-08), IPSN(04-07), MOBICOM(03-07), MOBIHOC(00-07), IMC(01-07) |
| Programming | 505 | POPL(02-08), PLDI(00-07), ICFP(02-07), OOPSLA(00-07) |
| Theory | 471 | SODA(01-08), STOC(00-07) |

To compare our performance with the state-of-the-art Web-based method, LiveClassifier [3], denoted as LC, has been implemented, where the concepts hierarchy are referred from DMOZ. For the CS-related classes, these concepts are "computers", "reference", "business", "software", and "science"; and for the classes of four universities, the concepts are "society", "people", "university", "school", "education", "sports", "United States". The thresholds $\delta$ and $t$ used in DM are set to 0.85 and 7 in both dataset, respectively. We use the Rainbow tool[6] and the VSM (Vector Space Model) classification model for all the experiments.

[5] The WebKB project: http://www.cs.cmu.edu/~WebKB/
[6] The Rainbow tool: http://www.cs.cmu.edu/~mccallum/bow/rainbow/

### 4.2 Text Classification

Table 5 compares the classification accuracy between different methods. For each class, the baseline method is to use only the class name as query, submit to the search engine, and collect snippets as training data. The method from query expansion (QE) is to use pseudo-relevance feedback (PRF) for each class, where the terms with high TF-IDF values in the snippets are selected as expansion terms which used for acquiring training data. LC is the method implemented based on LiveClassifier [3]. With the concept hierarchy of each class, all the concepts are used to combine with the class name as a query, then submit to the search engine and collect the snippets as training data.

From Table 5, we find that merely sending class names as queries cannot retrieve quality training data. It only achieves the accuracy at 0.57 on average. Even if we expand the class names with PRF (the method of QE), the accuracy cannot be improved by QE in this case. This is because PRF is a general solution to the keyword mismatching problem and thus would not be well applied to our classification problem. LC manually labels some concepts of the class names, e.g., the concept of "Architecture" is about computer architecture, so that more relevant training data to original classes can be fetched. LC gets higher accuracy at about 0.71. But such concepts labeled by people are often few. Our CM method can discover more useful common concepts, as shown in Table 6, where keywords "computers", "conference", and "proceeding" are all helpful in searching relevant data for each class when combined with original class name. CM, on the other hand, might introduce noisy keywords often co-occurring with the class names such as keyword "web". Due to the assistance of more common concepts, we collect more quality training data by CM. The accuracy for CM comes to 0.76, which is much better than LC and QE.

In addition, we observe that the performance for class "Network" is originally high in baseline method. To realize what makes the result, we check the process for getting training data in advance. Except for the accurate semantic for the term "Network", we discover that the snippets from search engine are suitable and with good quality for the class "Network". This is related to the characteristic of the Web. To our experience, the documents ranked from search engines might be mostly relevant to the fields about computer or network.

The results of Web pages classifications, as shown in Table 7, are similar to previous experimental results except the average accuracy obtained here is lower in general due to the noisy Web pages that are unreliable for testing. Moreover, sometimes the concepts derived from the Web are not effective in classification (even they are correct). For example, from the Web, DM learns two discriminative concepts of "ut" and "milwaukee" for classes "Texas" and "Wisconsin", respectively. But their impacts on classifying our testing data are futile. Although there are some noises in the Web page dataset and the concepts found are correct but less useful, CM+DM does an improvement while training by the snippets from these concepts, and surpasses the performance of LiveClassifier (LC). By our methods, the quality training data are fetched and the classifier learns better, thus become more robust for text classification.

Table 5. Accuracy of classification in CS papers.
Baseline: Class Names, QE: Query Expansion, LC:LiveClassifier,
CM: Common concept method, and DM: Discriminative concept method.

| **Method** | **Architecture** | **IR** | **Network** | **Programming** | **Theory** | **Avg.** |
|---|---|---|---|---|---|---|
| Baseline | 0.76 | 0.38 | 0.90 | 0.72 | 0.76 | 0.57 |
| QE | 0.03 | 0.01 | 0.98 | 0.71 | 0.66 | 0.48 |
| LC | 0.88 | 0.42 | 0.82 | 0.54 | 0.89 | 0.71 |
| CM | 0.80 | 0.77 | 0.89 | 0.56 | 0.76 | 0.76 |
| DM | 0.04 | 0.00 | 0.83 | 0.66 | 0.98 | 0.50 |
| **CM+DM** | **0.76** | **0.86** | **0.91** | **0.71** | **0.82** | **0.81** |

Table 6. Extracted common concepts and discriminative concepts of CS classes.

| Method | Architecture | IR | Network | Programming | Theory |
|---|---|---|---|---|---|
| CM | By DMOZ: **computers** By Star Algorithm: **conference, proceeding, web** | | | | |
| DM | architecture, architectural, architects, arts, history, contemporary, design | infrared, satellite, visible, image, weather, thermal, cameras | usa, health, action, sports, food, monitor, global | tutorial, java, example, using, oriented, download, linux | graph, mathematical, problems, literary studies, political, number, |
| CM+DM | isca, energy, oriented, adaptive, aidede, ecaade, architectural | investor, infrafed, retrieval, ecir, trec, sigir, ie | development , wireless, shows, server, first, email ,mail | ferment, oriented, programmers, final, siggraph, graphics, animation | Number, university, critical, math, computational, theoretical, complexity |

Table 7. Accuracy of classification in Web pages.
Baseline: Class Names, QE: Query Expansion, LC:LiveClassifier,
CM: Common concept method, and DM: Discriminative concept method.

| **Method** | **Cornell** | **Texas** | **Washington** | **Wisconsin** | **Avg.** |
|---|---|---|---|---|---|
| Baseline | 0.85 | 0.47 | 0.68 | 0.20 | 0.55 |
| QE | 0.87 | 0.40 | 0.71 | 0.41 | 0.60 |
| LC | 0.99 | 0.14 | 0.58 | 0.14 | 0.46 |
| CM | 0.99 | 0.30 | 0.72 | 0.33 | 0.59 |
| DM | 0.47 | 0.75 | 0.53 | 0.66 | 0.60 |
| **CM+DM** | **0.98** | **0.43** | **0.77** | **0.37** | **0.64** |

### 4.3 Combination of Web Training Data for Text Classification

In this experiment, we want to realize how our methods help conventional supervised text classification. We further conduct an experiment to compare the performance between labeled data plus training data from the Web and the labeled data only.

Both datasets are divided into training and testing data. We combine the training data collected from the Web and sample α % of the training data as a new training corpus. For comparison, we also train a classifier with just those α % of the labeled data only. The α value varies from 1 to 100. α =100 means to use all the labeled data from original divided training set, thus become a supervised learning. 5-fold cross validation is used to evaluate the classification accuracy.

Figure 1 and 2 show the two experimental results, respectively. We find that the Web corpus improves the classifier's accuracy more than 6% when α is small for both datasets. In other words, when the labeled data is insufficient or even with no quality, sampling training data from the Web could substantially complements the manually-labeled data. The performance of both classifiers increases when more labeled data are included. However, when more and more manually-labeled data is given, the improvement by the Web becomes less obvious or even slightly worse. This is because once we add more data from the Web, we also introduce the noises such that the accuracy glows slowly when the quality documents are enough. In CS papers classification, the performance reaches as higher as all training data used when 50% labeled data is added. For Web pages classification, we have the same performance as supervised learning when only use 20% labeled data. The performance even exceeds the result from supervised learning when 40% labeled data is joined. It explains more quality data from the Web is helpful in classification. This result also shows that using all the labeled data is not always as good as expected because some of the labeled data are not in good quality.

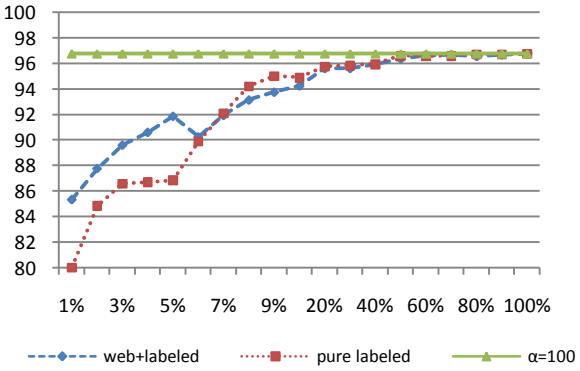


**Fig. 1.** Accuracy of training data from the Web plus % labeled data for CS papers classification.

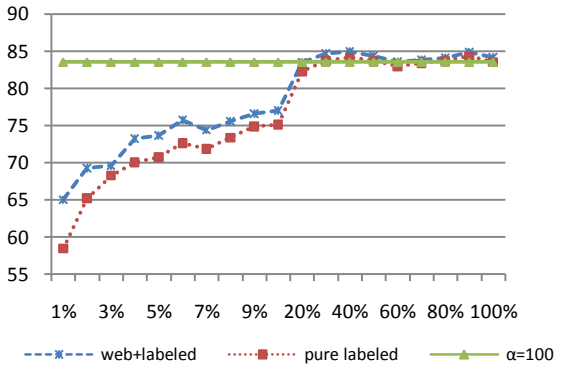


**Fig. 2.** Accuracy of training data from the Web plus % labeled data for Web pages classification.

This experiment shows us that the training data from the Web does help to improve the text classification. Moreover, we could use a very few number of labeled data, plus the Web corpus our methods collect, to train a desirable classifier. The Web helps the classifiers to learn the unseen concepts which do not exist due to the insufficiency or the unreliable quality of labeled data. It also tells us that the suitable training data might change by time, thus the original labeled data performs worse in new classification tasks. From the result, we believe that our methods can benefit the task of text classification, and other advanced applications.

## 5. Related Work

Text classification has been extensively studied for a long time in many research fields. Conventionally, supervised learning is usually applied in text classification [1, 2]. Our work focuses on the problem of how to adequately acquire and label documents automatically for classification models.

For the problem of automatic acquiring training data, previous studies discuss in two directions. One is focused on augmenting a small number of labeled training documents with a large pool of unlabeled documents [11, 12, 13, 14, 15, 16, 17, 18]. Such work trains an initial classifier to label the unlabeled documents and uses the newly-labeled data to retrain the classifier iteratively. [11] proposed by Nigam et al. use the EM clustering algorithm and the naive Bayes classifier to learn from labeled and unlabeled documents simultaneously. [12, 13] proposed by Yu et al. efficiently computes an accurate classification boundary of a class from positive and unlabeled data. In [15], Li et al. use positive and unlabeled data to train a classifier and solve the lack of labeled negative documents problem. Fung et al. in [17] study the problem of building a text classifier using positive examples and unlabeled example while the unlabeled examples are mixed with both positive and negative examples. [14] proposed by Nigam et al. starts from a small number of labeled data and employs a bootstrapping method to label the rest data, and then retrain the classifier. In [18], Shen et al. propose a method to use the n-multigram model to help the automatic text classification task. This model could automatically discover the latent semantic sequences contained in the document set of each category. Yu et al. in [16] present a framework, called positive example based learning (PEBL), for Web page classification which eliminates the need for manually collecting negative training examples in preprocessing. Although classifying unlabeled data is efficient, human effort is still involved in the beginning of the training process. In this paper, we propose an acquiring process of training data from the Web, which is fully automatic. The method trains a classifier well for document classification without labeled data, which is the mainly different part from the previous work. Moreover, our experiments show that the Web can help the conventional text classification. The training data acquired from the Web expand the coverage of classifier, which substantially enhance the performance while there is a lack of labeled data, or the quality of labeled data is not well enough.

Another direction is focused on gathering training data by the Web [3, 4, 5, 6]. In [3], Huang et al. propose a system, called “LiveClassifier”, which combines relevant class names as queries based on a user-defined topic hierarchy so that more relevant documents to the classes could be found from the Web. [4, 5, 6] proposed by Hung et al. presents an approach that assumes the search results initially returned from a class name are relevant to the class. These search results are treated as auto-labeled and additional associated terms with the class names are extracted from the labeled data. Although the previous works are similar to our methods, all of them are human-intervened. In this paper, we propose a method which automatically finds the associated concepts for the related classes and train a desirable classifier. The main contribution is that our method utilizes the relationship of classes and samples the Web in an automatic way for key concepts of each class, thus further find the

quality training data from the Web. Without labeled data and associated terms given by human, our methods perform well and classify documents accurately for the text classification problem.

## 6. Discussions and Conclusions

In this paper, we propose two methods to automatically sample the Web and find quality training data for text classification. We first examine the effects of different search engines, retrieved data types, and sizes of retrieved data. Moreover, from the subset of documents and the method by associated terms, we know that sampling the Web for concepts of classes and fetching training data can substantially improve the performance of classification. It might be hard to distinguish the classes with the ambiguity and close relationship without labeled data. By the discovering of common concepts and discriminative concepts, the ambiguity of class names is eliminated and more relevant concepts are utilized for sampling suitable and quality training data from the Web. Several experiments conducted in this work show that our methods are useful and robust for classifying documents and Web pages. Furthermore, our experiments show that the training data sampled from the Web helps the conventional supervised classification, which need quality and labeled data. The result demonstrates that the quality of labeled data might not always desirable due to the lack of useful key concepts, and we can provide proper training data from the Web to further improve the results of text classification. In additions, two dataset with different characteristics are used for our experiments and the analysis from different dataset is carefully conducted in this paper. Compared to previous works, the advantage of our methods is the fully automatic processes during the concepts expansion and the training data collecting. Our methods are independent of classification models, thus existing models can be incorporated with the proposed methods.

However, our work has some limitations. The classes we choose are related to each other. In other words, the performance would be better while the classes are in the same level in the hierarchy of topic classes. With the relationships between the classes, our methods can perform the context-aware technique among the classes to acquire more relevant documents, making the classifiers robust. To go a step further, there are more challenges to choose the quality documents in the training corpus sampled from the Web. We can also sample good training documents while a pool of unlabeled data is provided. We believe that these challenges are worth studied and would be the research directions in our future work.

## References

[1] Y. Yang and X. Liu, “A re-examination of text categorization methods,” in Proceedings of the 22nd Annual International ACM SIGIR conference, 1999, pp. 42–49.

[2] Y. Yang, “An evaluation of statistical approaches to text categorization,” Information Retrieval, vol. 1, pp. 69–90, 1999.

[3] C.-C. Huang, S.-L. Chuang, and L.-F. Chien, “Liveclassifier: Creating hierarchical text classifier through web corpora,” in World Wide Web Conference, 2004.

[4] C.-C. Huang, K.-M. Lin, and L.-F. Chien, "Automatic training corpora acquisition through web mining," in IEEE/WIC/ACM Conference on Web Intelligence, 2005.

[5] C.-M. Hung and L.-F. Chien, "Text classification using web corpora and em algorithms," in The Asia Information Retrieval Symposium, 2004, pp. 12–23.

[6] C. M. Hung and L. F. Chien, "Web-based text classication in the absence of manually labeled training documents," Journal of the American Society for Information Science and Technology, pp. 88–96, 2007.

[7] C. Carpineto, R. D. Mori, G. Romano, and B. Bigi, "An information theoretic approach to automatic query expansion," ACM Transactions on Information Systems, vol. 19(1), pp. 1–27, 2001.

[8] Y. Qui and H. Frei, "Concept based query expansion," in Proceedings of the 16th Annual International ACM SIGIR Conference, 1993, pp. 160–169.

[9] J. Xu and W. B. Croft, "Query expansion using local and global document analysis," in Proceedings of the 19th Annual International ACM SIGIR Conference, 1996, pp. 412–420.

[10] J. Aslam, K. Pelekhov, and D. Rus, "A practical clustering algorithm for static and dynamic information organization," in In: ACM-SIAM Symposium on Discrete Algorithms. In: ACM-SIAM Symposium on Discrete Algorithms (1999), 1999.

[11] K. Nigam, A. K. McCallum, S. Thrun, and T. M. Mitchell, "Text classification from labeled and unlabeled documents using em," Machine Learning, vol. 39(2/3), pp. 103–134, 2000.

[12] H. Yu, "Svmc: Single-class classification with support vector machines," in Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, 2003.

[13] H. Yu and C. Zhai, "Text classification from positive and unlabeled documents," in Proceedings of the 12th Annual International ACM Conference on Information and Knowledge Management, 2003, pp. 232–239.

[14] A. McCallum and K. Nigam, "Text classification by bootstrapping with keywords," in ACL Workshop for Unsupervised Learning in Natural Language Processing, 1999.

[15] X. Li and B. Liu, "Learning to classify texts using positive and unlabeled data," in Proceedings of International Joint Conferences on Artificial Intelligence, 2003.

[16] H. Yu, J. Han, and K.-C. Chang, "Pebl: Web page classification without negative examples," IEEE Transactions on Knowledge and Data Engineering, vol. 16, pp. 70–81, 2004.

[17] G. Fung, J. Yu, H. Lu, and P. Yu, "Text classification without labeled negative documents," in Proceedings of 21st International Conference on Data Engineering, 2005.

[18] D. Shen, J.-T. Sun, Q. Yang, H. Zhao, and Z. Chen, "Text classification improved through automatically extracted sequences," in Proceedings of the 22nd International Conference on Data Engineering, 2006.