Deep Classifier: Automatically Categorizing Search Results into Large-Scale Hierarchies

Dikan Xing[1], Gui-Rong Xue[1], Qiang Yang[2], Yong Yu[1]

[1]Shanghai Jiao Tong University, Shanghai, China
{xiaobao,grxue,yyu}@apex.sjtu.edu.cn

[2]Hong Kong University of Science and Technology, Hong Kong, China
qyang@cse.ust.hk

ABSTRACT

Organizing Web search results into hierarchical categories facilitates users' browsing through Web search results, especially for ambiguous queries where the potential results are mixed together. Previous methods on search result classification are usually based on pre-training a classification model on some fixed and shallow hierarchical categories, where only the top-two-level categories of a Web taxonomy is used. Such classification methods may be too coarse for users to browse, since most search results would be classified into only two or three shallow categories. Instead, a deep hierarchical classifier must provide many more categories. However, the performance of such classifiers is usually limited because their classification effectiveness can deteriorate rapidly at the third or fourth level of a hierarchy. In this paper, we propose a novel algorithm known as *Deep Classifier* to classify the search results into detailed hierarchical categories with higher effectiveness than previous approaches. Given the search results in response to a query, the algorithm first prunes a wide-ranged hierarchy into a narrow one with the help of some Web directories. Different strategies are proposed to select the training data by utilizing the hierarchical structures. Finally, a discriminative naïve Bayesian classifier is developed to perform efficient and effective classification. As a result, the algorithm can provide more meaningful and specific class labels for search result browsing than shallow style of classification. We conduct experiments to show that the Deep Classifier can achieve significant improvement over state-of-the-art algorithms. In addition, with sufficient off-line preparation, the efficiency of the proposed algorithm is suitable for online application.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous; I.5.4 [**Pattern Recognition**]: Applications—*Text processing*

WSDM'08, February 11–12, 2008, Palo Alto, California, USA.

General Terms

Algorithms, Experimentation

Keywords

Deep Classifier, Search Result Mining, Hierarchical Classification, Hierarchy Pruning

1. INTRODUCTION

With the increasing prevalence of Web technologies, Web search has become essential in everyday life. Current search engines typically return a long list of search results in response to a user-issued query. Although the most authoritive results may be ranked high among all results under a proper ranking algorithm (e.g. PageRank [20]), it remains a question whether the most authoritive pages are what the user actually wants. In particular, when a query is inherently ambiguous and the first pages are not the intended results of the users, sometimes users may have difficulty in subsequently browsing the rest of the pages. For example, if a user wants to find the benefits from an apple fruit and issues a query "apple", all results on the first page are focused on the topic about the computer company "Apple"[1], rather than the intended topic of the apple fruit. This is because the Apple company is so well-known that Web pages about the sense, that is the Apple company, are more likely to be well linked and thus be ranked higher. To make matters worse, the results about fruit apple are placed apart from each other in the list of search results. For the fruit apple, the results are placed at 14, 37, 38, 62, 63, 66, 67, 71, 75, 76, 82, 84, 88 among top 100 search results from the Google search engine.

A solution is to perform classification on the search results [5, 8]. As described in [5, 8], automatic category organization brings many advantages for users, where the search results are automatically compiled into a hierarchy according to different potential meanings. As pointed in these two papers, the users preferred the category interface much better than the list interface; in fact they were 50% faster in finding information that was organized into categories.

Generally, there are two types of models for classification: shallow classification and deep classification. Shallow classification [5, 8] trains the model on the top level or the top two levels of hierarchy while deep classification [15] learns

[1]http://www.apple.com/

the classification models on an entire large-scale hierarchy. However, the shallow classification scheme may be too coarse for the given search result data when all the resultant data items are placed in only two or three classes. In contrast, placing the search result on a category in a deep hierarchy can provide more content information on the results than with a shallow hierarchy. Among the previous researches, Liu et al. [15] introduced a top-down classification strategy on a deep but wide target hierarchy. The width leads to the large size of the hierarchy, resulting in a performance decline as the hierarchical depth increases. The measure of Macro-F1 was near 20% at the second level, and decreased to near 10% at the fifth.

In addition to an accuracy decrease, training the model on a global hierarchy may not be a good choice and yield poor performance. For example, the classification results of the query "apple" are supposed to be distributed among the categories such as *Health*, *Computer* (if these categories are available), while results of the query "Saturn" are expected to distribute among categories like *Science*, *Game* and *Car*. It is expected that a classifier into two categories *Health*, *Computer* will outperform a classifier into more categories such as *Health*, *Computer*, *Science*, *Game* and *Car* on search results in response to the query "apple". This implies that it is desirable to employ an adaptive method for creating classifiers that uses different target categories for different user queries.

According to the above observations, in this paper, we propose a novel algorithm known as *Deep Classifier* to classify search results into a large and deep target hierarchy *adaptively* by utilizing the existing hierarchies on the Web. We do this as follows. We first prune a large hierarchy into one with a smaller size while keeping the original hierarchical structure. This is accomplished by first querying an on-line Web directory for the specified query and then creating all ancestors of these deep categories. This results in a deep but *narrow* target hierarchy from the originally large and wide one. In this way, a different target hierarchy is created adaptively for a different user query. The leaf nodes in such a hierarchy will be taken as the category candidates for the search results categorization. Based on the narrow and deep hierarchy, we propose different strategies for training the data selection with the help of the hierarchical structure. These classification models are learned online using an efficient implementation of the discriminative naïve Bayesian classifier for classification.

Experimental results show that our Deep Classifier algorithm can achieve significant improvement over state-of-the-art algorithms. Furthermore, with sufficient off-line preprocessing, the efficiency of the proposed algorithm is suitable for online application. As a result, the entire algorithm can provide more meaningful and specific class labels for search result browsing when compared to the shallow counterparts.

It is worthy to mention that actually, the online Web directories such as ODP, Yahoo! Directory, etc, are straightforward solutions to what has mentioned above. Searching on these Web directories, users will be fed up with search results attached with category attributes, which has already been assigned by human editors and are of course hierarchical. However, the obvious deficiency of these solutions is that the number of assembled Web pages is limited in any human maintained Web directories. Pages one can search from these web directories are much less than those from a search engine. Our solution combines Web directories and search engines. From the former, we gain a rich (large and deep) concept hierarchy and from the latter, we are able to search among billions of pages indexed by the search engine.

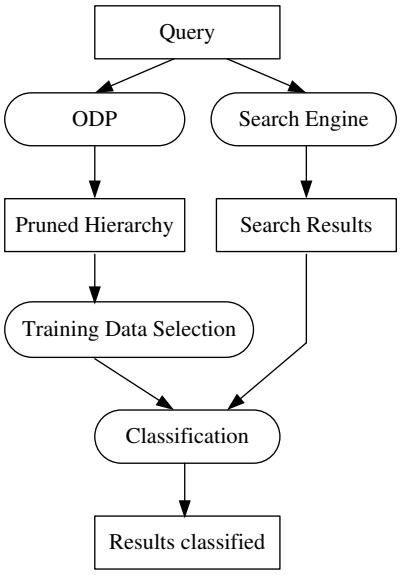

Figure 1: Overview of Deep Classifier

The remainder of this paper is organized as follows. Section 2 gives a brief overview of the Deep Classifier algorithm. In Section 3, we propose different strategies for training data selection. In Section 4, a discriminative naïve Bayesian classifier is presented. The implementation issues are described in Section 5. We will report and analyze the results of a series of experiments of our proposed algorithm in Section 6. Related work is discussed in Section 7. In Section 8, we will give a conclusion and discuss the future work.

2. OVERVIEW OF DEEP CLASSIFIER

In this section, we give an overview of our Deep Classifier algorithm. Figure 1 illustrates the flowchart of our system. We first let a user issue a query, which will be submitted to an online Web directory (e.g. Open Directory Project (ODP)[2]) and a search engine (e.g. Google[3]) simultaneously to get the category information and search results, respectively.

The online Web directory may respond a list of categories that are relevant to the query. For example, by issuing the query 'Saturn' to ODP, ODP will return five categories (categories with bold font in the right part of the Figure 2). By creating the pruned hierarchy from the five categories, only twenty-four ancestors remain. Compared with the entire hierarchy as shown to the left of the Figure 2, this narrow-down procedure helps us greatly reduce the number of target category candidates.

The leaf nodes in the pruned hierarchy are regarded as our target category candidates. These nodes may have offspring in the original large hierarchy but offspring are now pruned. In other words, we may also pick up internal nodes, instead

[2]http://dmoz.org
[3]http://www.google.com

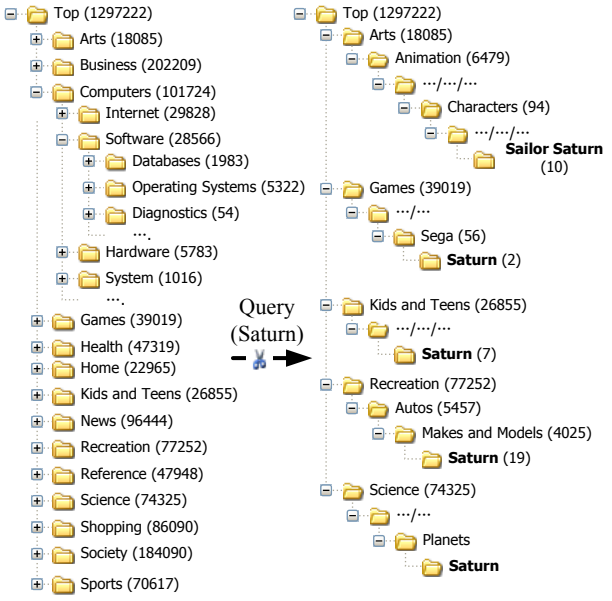

Figure 2: Prune Large and Deep Hierarchy into Decent Size.

of only leaf nodes, of the original large hierarchy as our target category candidates.

Next, based on the structure of the pruned hierarchy, three training data selection strategies are proposed in Section 3 by utilizing the hierarchical structure. This step is important since the labeled Web pages under one ODP category are usually too few to train a reliable classifier.

Then, based on the training data, we perform classification model learning. Since our algorithm is conducted online, it is important for the algorithm to be efficient. To satisfy this goal, we propose a simple classifier based on naïve Bayesian which is described in Section 4. We will also compare our results with the Support Vector Machine in experiments to see how much time one should wait to obtain a probably better classification results by SVM.

Finally, we classify the search results and demonstrate the results with a hierarchical search result interface for the user. Compared with a top-level categorization or a top-two-level categorization proposed in [5, 8] like

Arts
Games
Kids and Teens
Recreation
Science

Arts
Animation
Game
Video Game
Kids and Teens
School Time
Recreation
Autos
Science
Astronomy

our proposed Deep Classifier can (1) provide an interface with more meaningful class labels shown in the right part of Figure 2 than top-or-top-two-level style and (2) classify search results more delicately. The second point is not very obvious in this case, since the number of category candidates does not change between shallow and deep classification, but we will show examples of this kind in the experiment section later (in Table 1).

3. TRAINING DATA SELECTION

In this section, we discuss strategies for training data selection. This is a very important issue in our task. In Figure 2, the number after category name is the number of training data attached to the category (and its offspring). One can see that some category candidate contains very few labeled Web pages, which is insufficient to build a reliable classifier. We will propose three different strategies for selecting the training data for our task. The best strategy is used in our final design of the Deep Classifier.

3.1 Flat Strategy

The flat strategy is a simple method for training data selection, in which we transfer the hierarchical classification task into a flat classification task. From the viewpoint of hierarchical classification, this strategy places all the category candidates (those in bold font in Figure 3) directly at the root, which is shown in Figure 4. Then, we directly classify the search results into category candidates by a flat classifier. As shown in Figure 4, we can directly train the model using the data from categories 44, 85, 205, 66, 874, 902, 42, 677 and 707. This method is simple to use, but it does not consider the hierarchical structure of web directory.

3.2 Hierarchical Strategy

Since a hierarchy is pruned to a manageable size, the existing top-down style can be tried even for online application.

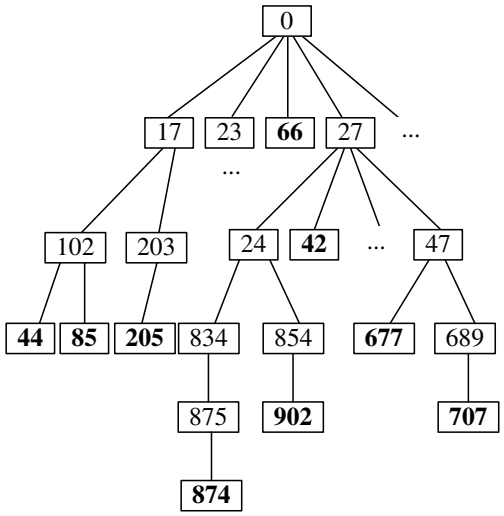

Figure 3: An Example of Pruned Hierarchy

We now describe the algorithm according to the tree structure shown in Figure 3. The category candidates are shown in bold font, and the hierarchical strategy is to first classify (estimate probabilities of) the search results into categories marked with 17, 23, 66 and 27. Thus a classifier is created by selecting training data under the nodes (and their offspring) 17, 23, 66 and 27. The estimated probability of each non-candidate category is propagated to their candidate offspring. The rest of the algorithm is similar to that at the

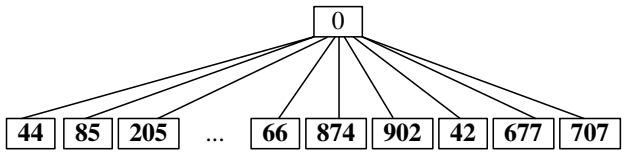

Figure 4: Flat Strategy

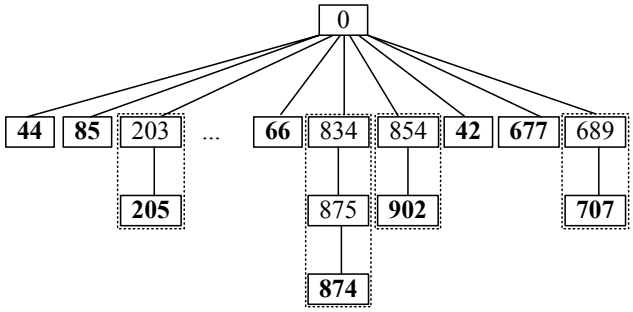

Figure 5: Ancestor-Assistant Strategy

top level. For example, another classifier will be created by selecting the training data from the categories 102 and 203 and then classifying the search results into them.

Formally, we represent the category candidate c_i with its ancestors, denoted by $c_i^1, c_i^2, \cdots, c_i^{l_i}$ where l_i is the length of the path from the category candidate c_i to the root. c_i^1 is the root node and $c_i^{l_i}$ is the category candidate itself. The probability $P(c_i^1|x)$ equals to 1. Now we formalize the probability $P(c_i|x)$ that the test case x belongs to c_i by:

$$\begin{aligned} P(c_i|x) &= P(c_i^1, c_i^2, \cdots, c_i^{l_i}|x) \\ &= P(c_i^{l_i}|x, c_i^1, c_i^2, \cdots, c_i^{l_i-1}) \times P(c_i^1, \cdots, c_i^{l_i-1}|x) \\ &= \cdots \\ &= \prod_{k=1}^{l_i} P(c_i^k|x, c_i^1, \cdots, c_i^{k-1}) \end{aligned}$$

Each conditional probability in the product is estimated by a classifier at the k^{th} level under the category c_i^k. If there is only one child under c_i^k in the pruned tree, 1 is assigned directly without the cost of classifier training. Search results are classified by finding c_i which maximize the *a posterior* $P(c_i|x)$.

This strategy is different from other two since it requires learning more than one classifiers before making a final decision on classification.

Several state-of-art algorithms worked in a similar hierarchical way [5, 8, 15, 3]. Presenting this strategy avails us to make comparison with such work.

3.3 Ancestor-Assistant Strategy

The ancestor assistant strategy is guided by the following two considerations. First, the training data from the category candidate itself may be insufficient in size, especially for a deep category. Thus, we need to obtain more data elsewhere. Second, the training data from its "high" ancestors may be too general to reflect the characteristics of the deep category candidate. In one word, we want to some borrow training data from the ancestors but should not go too high up.

Hence, we propose a trade-off between the hierarchical strategy and flat strategy. That is, we combine the training data from the category candidate itself and those from its ancestors and siblings. More precisely, we find the farthest ancestor of the category candidate which is not the ancestors of other category candidates. And the Web pages at this ancestor and all offspring of this ancestor are picked up.

The training data for category 874 are from those of 834 and all offspring of 834 while the training data for category 902 are from those of 854 and all offspring of 854 since the common ancestor is category 24 (in Figure 3).

From the viewpoint of hierarchical classification, we attach the maximal subtree containing one and only one category candidate to the root. The tree in Figure 5 can clearly clarify this strategy.

4. CLASSIFICATION MODEL

Due to the adaptive nature of the problem, a classifier that is fast to learn is preferred because the classifiers are trained online at query time. If a classifier such as SVM is employed, the long training time might prevent us from delivering the result in a timely manner. To this end, we prefer the naïve Bayesian classifier.

4.1 Event Model

There are different event models for inputs of naïve Bayesian text classifier. [16] mentioned two: multi-variate Bernoulli and multinomial models. These two models regard each document as a whole and the joint distribution of the whole documents and the target categories are considered. In this paper, we use the event model introduced in [17], which makes it easy to interpret our later modification on model.

We regard each document as a sequence of random variables, $A_1, \ldots A_n$, where n is the length of the sequence (document). Each A_i corresponds to the i^{th} word in the document. These random variables are independently, identically distributed, as the "naïve" assumption says. This makes observation on any position in a document can be used to estimate for all positions. The support of these variables are all distinct word IDs. In later discussion, we do not explicitly attach a position subscript for A.

Then we do not directly model the joint distribution between documents and categories but between words and categories. And a document is the intersection of a sequence of word events. Or in other words, if one has a N-faced dice where N denotes the vocabulary size and toss it up for finite times, this process generates a document by recording each result after tossing up the dice.

4.2 Naive Bayesian Classifier

Under the above event model, the classifier actually estimates $P(A = w_j|c_i)$ for all i and j where j iterates over all words and i iterates over all target candidates. The classifier estimates the probability that a document (sequence of word events) belongs to a category by computing

$$\begin{aligned} P(c_i \mid v) &\propto P(v \mid c_i)P(c_i) \\ &= P(c_i)\prod_{j=1}^{N} P(A = w_j \mid c_i)^{v_j} \end{aligned} \quad (1)$$

where c_i is a category, v is the document, N is the vocabu-

lary size, w_j is each word in vocabulary, and v_j is the corresponding count in v for word w_j.

$P(c_i)$ is called the *a priori* probability that a document v belongs to category c_i and usually estimated by counting training data from different categories. However, in our setting, the training data are taken from the manually created hierarchies such as ODP, and the test examples are from search engines such as Google. The obtained top 100 (or 200) search results (test cases) may be incomplete. That means the distribution of categories may not be the same as that in a manually created hierarchy. We observed this inconsistency in most of our evaluated queries. Therefore, we believe that it is preferred to weaken the *a priori* probabilities and remove the $P(c_i)$ item in the formulation. This can also be regarded as the assumption that the *a prior* probability of each category is weakened to the same amount, $1/n$, where n is the number of categories, according to the maximum entropy principle. Besides, [12] assumed that *a priori* did not actually have great impact.

4.3 Discriminative Naive Bayes Classifier

Applying Bayes' theorem, we obtain

$$\begin{aligned} P(A = w_j|c_i) &= \frac{P(A = w_j,\ c_i)}{P(c_i)} \\ &= \frac{P(c_i|A = w_j)P(A = w_j)}{P(c_i)}. \end{aligned}$$

Under the assumption that all $P(c_i)$ are equal, we can rewrite

$$\begin{aligned} & \prod_{j=1}^{N} P(A = w_j|c_i)^{v_j} \\ = & \prod_{j=1}^{N} \left(\frac{P(c_i|A = w_j)P(A = w_j)}{P(c_i)} \right)^{v_j} \\ = & \prod_{j=1}^{N} P(c_i|A = w_j)^{v_j} \times \prod_{j=1}^{N} \left(\frac{P(A = w_j)}{P(c_i)} \right)^{v_j} \\ \propto & \prod_{j=1}^{N} P(c_i|A = w_j)^{v_j} \end{aligned}$$

since all $P(A = w_j)$ are identical across different categories given the document. Now we arrive at

$$P(c_i \mid v) \propto \prod_{j=1}^{N} P(c_i|A = w_j)^{v_j} \quad (2)$$

This form facilitates another way of parameter estimates.

From the form of (1), each item $P(A = w_j|c_i)^{v_j}$ in the score function indicates that if w_j is common in a category c_i, then a test case with high occurrence of w_j will receive a high score. Likewise, if w_j occurs very little in a category c_i, then a test case with low occurrence of w_j will receive little penalty from the occurrence of w_j. Thus, we can regard each $P(A = w_j|c_i)$ as a vote on similarity between the category c_i and the test case v. The ultimate score is the combination of votes on *similarity* from all occurring words.

From the form of (2), we can regard each word as a vote on *discrimination*. If one word only occurs in one category, which can be regarded as a discriminative word for that category, it will contribute a maximum vote, 1, regardless of how many other words occur in that category, which may influence or *weaken* the vote from this word in the standard naïve Bayesian classifier discussed in Section 4.2. Therefore, we refer to this classifier as discriminative naïve Bayesian classifier.

4.4 Parameter Estimate

If we denote the occurrences of the event $(A = w_j, c_i)$ in training data as δ_{ij}, for (1), a maximum likelihood estimate yields

$$P(A = w_j|c_i) = \frac{\delta_{ij}}{\sum_{j'} \delta_{ij'}}$$

and for (2),

$$P(c_i|A = w_j) = \frac{\delta_{ij}}{\sum_{i'} \delta_{i'j}}$$

So, for standard naïve Bayesian classifier, the discriminant function for c_i is

$$f_i(v) = \sum_j v_j \ln \frac{\delta_{ij} + \epsilon}{\sum_{j'} (\delta_{ij'} + \epsilon)}$$

and for discriminative naïve Bayesian classifier, the discriminant function for c_i is

$$g_i(v) = \sum_j v_j \ln \frac{\delta_{ij} + \epsilon}{\sum_{i'} (\delta_{i'j} + \epsilon)}$$

ϵ is used for smoothing.

5. IMPLEMENTATION

We employ the ODP hierarchy in this work. This directory contains 17 top-level categories such as Business, Computer, Game, Health, etc. There are totally 712,548 categories and 4,800,870 web pages in our dumped version.

To reduce the scale of whole hierarchy to a manageable size and limit ourselves to English content only, three top-level categories, *Adult*, *Regional* and *World* are not used in our whole system. If the category candidates under these three categories are returned from ODP after a query, we just ignore them. Thus, $1,946,361$ documents and $170,198$ categories are remained.

5.1 Off-line Cache

To create the classifier efficiently given a set of categories, we first try to download all registered Web pages in ODP. The Web pages crawled are somewhat less than those registered in the ODP hierarchy since some network errors occurred during crawling. In total we crawled $1,297,222$ Web pages that are distributed over $157,927$ categories.

Then, we can build a vocabulary list of all Web pages crawled. The original vocabulary contains 6,387,537 distinct words[4]. We perform a global feature selection in which words with low occurrence are removed. Words less than 4 occurrences over the whole hierarchy are removed from our vocabulary. This reduces vocabulary by 80%, down to 1,287,715 words.

We pre-counted and saved δ_{ij} for all words in all categories. Thus a classifier can be quickly created by reading these δ_{ij} for involved words in involved categories.

We use the Ancestor-Assistant strategy and discriminative naïve Bayesian classifier in our final design of the Deep Classifier, which will be shown best among other alternatives later.

[4] "word" here means a sequence of letter, number and hyphen and not starting and ending with a hyphen

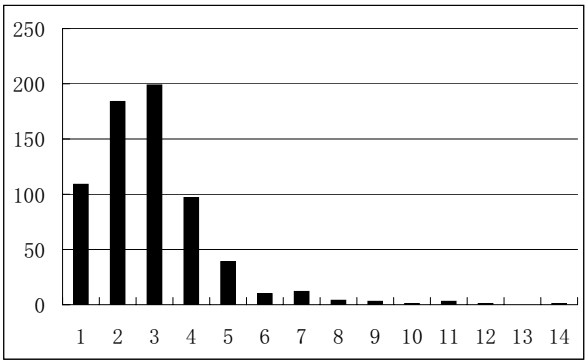

Figure 6: # of Queries vs. # of Top-level Categories

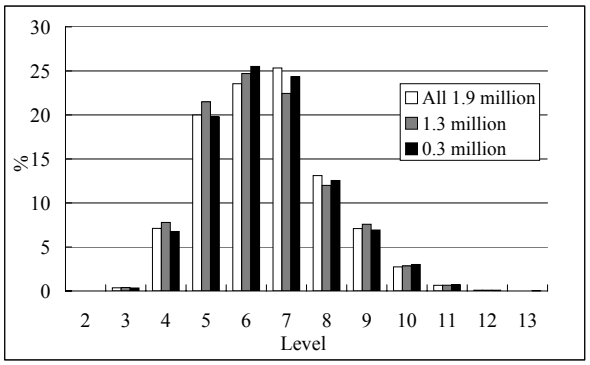

Figure 7: Distribution of Document under Different Levels

5.2 Space and Time Complexity

The Deep Classifier requires a two-dimension table of δ_{ij}. Each row corresponds to a category and each column a word in the vocabulary. The space complexity is $O(M*N)$, where N is the size of vocabulary and M is the number of all categories.

Not all the columns are loaded at initialization for space consideration. The *Second Chance* strategy [22] is employed to swap in and out rows.

When testing a search result, only words occur in the snippets are considered. So the time complexity for testing is $O(n * logN + n * m + K)$, where n is the length of the snippets, m is the number of category candidates and N is the size of the whole vocabulary. The first item denotes the time to convert snippets into word IDs. The second item denotes the time to classify. And K is the time for memory swapping only when the required column has not been loaded previously, so it is optional.

6. EXPERIMENTS

In this section, we first show the collected statistics about the category distribution for queries. Then we introduce the data sets we used for evaluation. And then we validate on Dataset I our statement in Section 3 that training data is usually insufficient at deep categories. Finally, performances of different strategies for collecting training data proposed in Section 3 and different classifier models proposed in Section 4 are compared on the two datasets.

6.1 Category Distribution

We have reported in the Introduction section that the categories of a query are usually distributed among a few concentrated categories. We collected 1000 popular queries from a famous search engine, and counted how many top-level categories the results of these queries from ODP are distributed over. The resultant distribution is shown in Figure 6. There are about 94.7% queries are distributed over less than six categories and about 74.2% queries are over three or less categories. The two most widely distributed queries are *games* (in 14 top-level categories) and *books* (in 12 top-level categories).

This justifies two issues: (1) Directly classifying search results into top-level categories may be too coarse for many queries and arranging search results into deep categories can separate them into more different categories; (2) Adaptively learning a classifier on different categories for different queries is preferred to a universal counterpart on all categories since queries will not be distributed over too many categories. This is obvious especially at the top level.

6.2 Data Set

To the best of our knowledge, there is no standard data set for our evaluation. In order to evaluate the performance of the Deep Classifier, we prepared two datasets, a large one and a small one, for evaluation. In each dataset, we collected top 100 search results for each query.

6.2.1 Dataset I: Randomly Selected 100 Queries

The ODP data we used in our Deep Classifier implementation (discussed in Section 5) for classifier training is about 1.3 million pages, there are another about 0.6 million pages are not included. We use half [5] of them to build a search engine indexed by Lucene[6]. This engine was built to simulate the role of search engine like Google.

We randomly selected about 100 queries from query logs and collected top 100 search results of each query from our "faked" search engine. Since each search result is from the ODP, the category attribute is already present. This greatly saved human labor to annotate large number of search results with categories and made it possible to prepare such a large dataset in limited time and human labor.

We also compared the distribution of the three collections: full 1.9 million pages registered in ODP, 1.3 million pages mentioned in Section 5 for Deep Classifier training and 0.3 million pages for the "faked" search engine. Both the percentage of document number and category number under each level are close in these three collections. The comparison of document number is shown in Figure 7. That of category number is similar. This ensures that we will not introduce great bias of search results towards some levels, deep or shallow, of categories. Likewise, this also ensures that the network failure mentioned in Section 5 did not cause a greatly inconsistent distribution of both Web pages and categories with the full version.

On this dataset, we also validated the assumption that

[5]The 0.6 million pages are those which cannot be downloaded at the first time and this half is those successfully downloaded at a second try.

[6]http://lucene.apache.org/

Table 1: Selected Ambiguous Queries

query	# of cat	# of top-level cat	max depth
ajax	3	2	5
apple	17	5	5
dell	6	3	6
jaguar	6	5	5
java	20	4	5
saturn	5	5	10
subway	4	2	5
trec	4	3	4
ups	4	3	5

training data at deep categories are very few. Flat strategy only obtained 21.6 training examples per query and category candidate. This figure significantly increased to 661.2 when the Ancestor-Assistant strategy is employed.

6.2.2 Dataset II: Ambiguous Queries

Ambiguous queries are limited in number and queries randomly selected in the manner described above are mostly unambiguous. So the first dataset can only reflect the performance of our system on facet categorization of unambiguous queries. To make up this deficiency, we asked our users to label the top-100 search results from real Google for 9 selected ambiguous queries. Each search result was labeled by two users and if the category they labeled did not agree, we just ignore that search result for that query.

Table 1 lists all the queries used in our evaluation system. For each query, we also list the number of category candidates returned from ODP, the number of different top-level categories, and the maximal depths of category candidates. The different values between the second and third columns supports the statement claimed in Section 2 that classifying on large and deep hierarchy can be more delicate.

6.3 Overall Performance

In this section, we will introduce the experiments we have conducted to validate these three hypotheses:

1. The Ancestor-Assistant strategy may outperform the hierarchical strategy and the flat strategy.
2. The discriminative naïve Bayesian classifier may outperform the standard one.
3. The discriminative naïve Bayesian classifier is supposed to be much faster than SVM although some performance may be lost.

6.3.1 Strategies for Selecting Training Data

In this experiment, we fix to employ the discriminative naïve Bayesian classifier, which will be shown to achieve higher performance later, and vary the strategies for selecting the training data. The Micro-F1, Macro-Precision, Macro-Recall and Macro-F1[23] on Dataset I are reported in Figures 8, which are averaged over all queries. For each individual query in Dataset II, we report these measures separately in Table 2.

As shown, we can find the Ancestor-Assistant strategy for training data selection can achieve highest performance. There are about 11.3%, 3.4%, 13.4%, and 15.7% improvement on Dataset I and about 53.3%, 52.4%, 43.1%, and

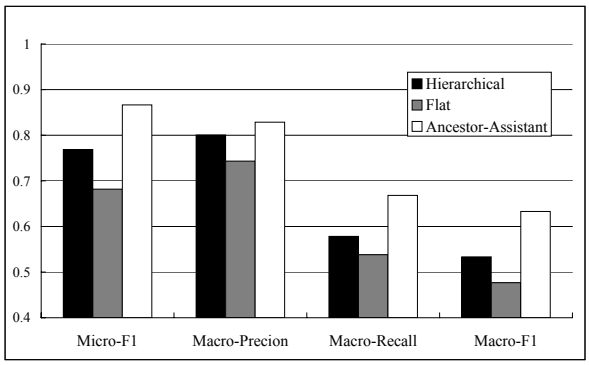

Figure 8: Dataset I: Different Strategy for Training Data Selection

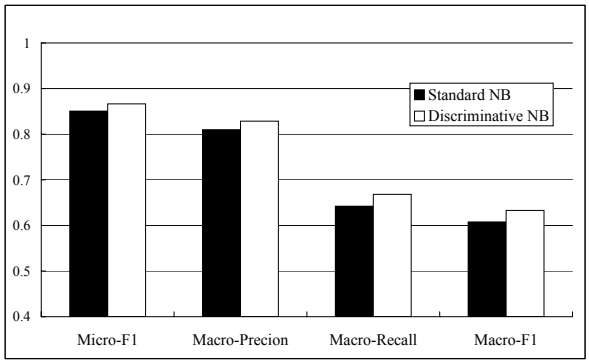

Figure 9: Dataset I: Different Classifiers

60.0% improvement on Dataset II from the hierarchical strategy on the measures of Micro-F1, Macro-Precision, Macro-Recall, and Macro-F1, respectively. There are about 21.3%, 10.3%, 19.5% and 24.7% improvement on Dataset I and about 14.1%, 14.3%, 68.8%, and 67.7% improvement on Dataset II from the flat strategy on these measures. The performance of the proposed Ancestor-Assistant strategy improved significantly from both the hierarchical and flat strategies. This validates the first hypothesis that the Ancestor-Assistant strategy which employs the training data from both the category candidate itself and its ancestors and siblings, and converts hierarchical classification into flat mode is the best.

One can find that the flat strategy is not stable. When the training data from the category candidate itself is very insufficient, the performance will be very poor, especially on Macro-Recall. But if sufficient, it achieves comparable performance to the Ancestor-Assistant strategy.

We consider the low performance of the hierarchical strategy which is similar to what is adopted in some existing work [8, 5, 15, 3] as due to the following two factors:

- The top-down scheme will accumulate the error rates at each level which gradually reaches an unbearable amount at some deep level of the hierarchy. This problem is overcome in our flat and Ancestor-Assistant strategies where the classification is performed using a flat mode.

Query	Micro-F1			Macro-Precision			Macro-Recall			Macro-F1		
	flat	hie	aa	flat	hie	aa	flat	hie	aa	flat	hie	aa
ajax	0.99	0.86	0.99	**0.92**	0.52	0.85	0.67	0.62	0.67	0.64	0.37	0.64
apple	**0.77**	0.21	0.72	**0.74**	0.28	0.68	0.33	0.26	**0.52**	0.29	0.19	**0.50**
dell	**0.67**	0.62	0.64	0.20	0.35	**0.39**	0.23	0.30	**0.54**	0.18	0.30	**0.33**
jaguar	0.61	0.41	**0.94**	0.59	0.51	**0.83**	0.30	0.53	**0.85**	0.26	0.45	**0.83**
java	**0.93**	0.29	0.83	0.48	0.34	**0.62**	0.37	0.20	**0.58**	0.32	0.20	**0.49**
saturn	0.71	0.41	**0.98**	0.60	0.69	**0.76**	0.43	0.63	**0.98**	0.29	0.50	**0.79**
subway	0.91	0.86	**0.94**	0.70	0.44	**0.71**	0.69	0.54	**0.88**	0.67	0.47	**0.78**
trec	0.60	0.52	**0.80**	0.34	0.34	**0.61**	0.54	0.54	0.46	0.38	0.38	**0.44**
ups	0.72	0.81	0.81	0.32	0.31	**0.33**	0.36	0.72	0.72	0.24	0.41	**0.43**
(average)	0.78	0.55	**0.85**	0.56	0.42	**0.64**	0.41	0.48	**0.69**	0.35	0.36	**0.58**

Table 2: Dataset II: Different Strategies for Training Data Selection

Query	Micro-F1		Macro-Precision		Macro-Recall		Macro-F1	
	std	dis	std	dis	std	dis	std	dis
ajax	0.98	**0.99**	0.68	**0.85**	0.64	**0.67**	0.64	0.64
apple	0.56	**0.76**	0.40	**0.68**	0.46	**0.52**	0.33	**0.50**
dell	0.66	**0.68**	**0.40**	0.39	0.23	**0.54**	0.20	**0.33**
jaguar	0.95	0.95	0.70	**0.83**	0.85	0.85	0.74	**0.83**
java	0.87	**0.90**	0.46	**0.62**	0.54	**0.58**	0.41	**0.49**
saturn	0.95	**0.99**	0.69	**0.76**	0.95	**0.98**	0.75	**0.79**
subway	0.86	**0.94**	0.56	**0.71**	0.83	**0.88**	0.65	**0.78**
trec	0.80	**0.82**	0.26	**0.61**	0.36	**0.46**	0.25	**0.44**
ups	0.75	**0.81**	0.33	0.33	0.38	**0.72**	0.26	**0.43**
(average)	0.82	**0.87**	0.50	**0.64**	0.55	**0.69**	0.47	**0.58**

Table 3: Dataset II: Different Classifiers

- In TAPER[3], the lengths of all paths from the root to the leaves equal, but this is not true for general. The probability estimated by this strategy is the product of different number of conditional probabilities, which necessarily favors those leaves at shallow levels.

6.3.2 *Standard vs. Discriminative Naive Bayesian Classifier*

In this section, we conduct experiments to validate the second hypothesis.

Since the Ancestor-Assistant strategy is validated the best, we only report the subsequent experiments on this strategy. We compared the performance between the standard naïve Bayesian classifier and the proposed discriminative naïve Bayesian classifier. The results are shown in Figure 9 for Dataset I, and Table 3 for Dataset II.

The proposed discriminative naïve Bayesian classifier can achieve about 2.8%, 2.3%, 3.9% and 4.0% improvement on Dataset I and about 7.3%, 28.8%, 25.6% and 24.0% improvement on Dataset II from the standard naïve Bayesian classifier on the measures of Micro-F1, Macro-Precision, Macro-Recall, and Macro-F1, respectively.

We consider the higher performance produced by the discriminative naïve Bayesian classifier is based on following observations. When two categories in deep level of the hierarchy need to be distinguished and they have a common ancestor in deep level of hierarchy too, e.g. the category 874 and 902 in Figure 3 and their common ancestor 24, these two categories consist many common words in major contents and only a few discriminative words that are of competence to discriminate these two categories. In such situation, in standard naïve Bayesian classifier, the contribution from these discriminative words are shrunken by a large denominator while discriminative naïve Bayesian classifier will assign higher weight for these discriminative words.

The experimental result also validates the second hypothesis that the discriminative naïve Bayesian classifier outperforms the standard one.

6.3.3 *Naive Bayesian Classifier vs. SVM*

We also compared naïve Bayesian classifiers with the sophisticated SVM on Dataset I to validate our third hypothesis. We use the libsvm package[4] for SVM implementation. It employs one-against-rest strategy to support multi-

classification. We failed to observe statistically significant improvements by SVM, but the time used by SVM was about 20 times of the time used by naïve Bayesian classifiers. The time recorded for SVM included the time to fetch training examples in vector forms via disk I/O, model training and classification. For each query, we clear the memory cache before classification, so the time for naïve Bayesian classifiers also included I/O cost for fetching saved δ_{ij}. In other words, this is the maximum time needed by naïve Bayesian classifiers. If the time for waiting the response from ODP and the search engine are excluded, the time used by discriminative naïve Bayesian classifier is averagely less than 1 second including any optional I/O cost, which makes it suitable for online application.

6.3.4 Significance Test

We perform one-tailed paired t-tests to check whether improvements mentioned above are of statistical significance. The p-values are shown in Table 4.

Dataset	Pair	Micro-F1	Macro-Precion	Macro-Recall	Macro-F1
I	dis vs std	0.00073	0.00402	0.00289	0.00372
	aa vs hie	0.00000	0.03303	0.00000	0.00000
	aa vs flat	0.00000	0.00039	0.00000	0.00000
II	dis vs std	0.01941	0.00289	0.01943	0.00040
	aa vs hie	0.00290	0.00072	0.00343	0.00077
	aa vs flat	0.03711	0.03383	0.00093	0.00228

Table 4: P-values of T-tests

7. RELATED WORK

In this section, we discuss some main researches related to our problem, including search result categorization, classification on hierarchical categories, Web directory based applications and classification models.

7.1 Search Result Categorization and Clustering

As a retrieval system though, TAPER[3] also included search result classification. The classification was performed off-line rather than at query time. The standard naïve Bayesian classifier combined with the hierarchical strategy for training data selection proposed in Section 3.2 is very similar to the classification model employed in TAPER[3].

Search result classification algorithms have been proposed in [5, 8], which corresponds to the shallow classification algorithms in this work. Dumais and Chen [8] have developed a system called SWISH in which the search results were automatically categorized based on the top-two-level categories of the LookSmart directory, where Web pages in the same category were clustered together. As described in their work, the category organization brings many significant advantages for Web users. Participants liked the category interface much better than the list interface, and they were 50% faster in finding information that was organized into categories. In [13], researchers used the top-level ODP categories as the training examples. For the end user, the most notable consequence of using a classification technique is the quality of the category names. Because the documents are classified to an existing taxonomy, the class names are predefined and can be carefully selected to optimally convey the intended meanings. Thus the naming problem associated with the clustering methods is avoided. However, the classification scheme may be too coarse for the given data set resulting in a top-level categorization where all data items are placed in one or two classes.

Clustering is a way to show the results in more detail topics. Many algorithms on search result clustering are proposed, e.g. Vivisimo[7]. As mentioned in [11], besides the lack of quality of the resulting clusters, the disadvantages include the lack of predictability of the clustering results and the diverse mix of the obtained sub-cluster hierarchies. This problem is even there for state-of-the-art search engine systems such as Vivisimo.

7.2 Classification on Hierarchical Categories

Several other researches have investigated the problem of classification over hierarchical taxonomies [2, 10, 21, 24]. Most of their findings over the testing data sets only numbered in hundreds, or at most a few thousand categories. Liu et al. [15] conducted a large scale analysis on the entire Yahoo categories to show the performance of classification on a large scale taxonomy. As stated in that paper, there are about 246,279 categories in Yahoo! Directory. The performance of classification on the top-level categories is about 72% of the Micro-F1 measure. However, when classifying the documents into deeper categories, the performance decreased quickly. As shown in [15], the performance is about 30% lower on the measure of Micro-F1 at the 4th level and deeper. Directly building large scale taxonomy classifiers cannot work in solving this problem due to the poor performance of classification.

7.3 Web Directory Application

Web directories, such as Yahoo!, Open Directory Project (ODP), Gimpsy and others, asked human editors to review Web sites under the Web directories. The most notable Web directory is the Open Directory Project, which is one of the largest efforts to manually annotate Web pages. Over 70,430 editors are engaged in keeping the directory reasonably up to date. As a result, the ODP now provides access to over 4 million Web pages in the catalogs. These Web directories are organized in a tree structure. In addition, each category has been labeled with an understandable name. The ODP has been successfully utilized in much related work, such as classification [8, 5, 18], personalized Web search [6, 14], evaluation [9, 1], etc.

7.4 Classifier Model

The naïve Bayesian classifier has been used for a few decades. Serval work (e.g.[7]) tried to explain its success although the naïve assumption usually does not hold.

Although the names share the same word "discriminative", our discriminative naïve Bayesian classifier is different from discriminative models discussed in [19], which compared the discriminative classifier and the generative classifiers. Our methods focused on discriminative "features" rather than the "classifier" itself.

The reversed naïve Bayesian classifier mentioned in [12] is very similar to our discriminative naïve Bayesian classifier. But the event model are different, and we also proposed a discrimination explanation which was not presented in [12].

[7]http://vivisimo.com

In that paper, the author claimed that the reversed version and the original version is the same under the assumption that there are the same number of training data in each categories, which is not true of our situation.

8. CONCLUSIONS AND FUTURE WORK

The existing algorithms for search result classification are based on shallow levels of topic hierarchies. Therefore, they are too coarse to provide the needed information for user browsing. The classification algorithms on large and deep hierarchies can provide much detailed categories but traditionally have been done with low classification performance. In this paper, we proposed a novel algorithm known as Deep Classifier to classify search results into deep topic hierarchies in an adaptive manner. By pruning the original hierarchy into a decent size while reserving the hierarchical structure and depth, we introduced different strategies for collecting sufficient training data in order to build reliable classifiers. Discriminative naïve Bayesian classifiers are employed for efficiency and effectiveness. Experimental results showed that the performance of the Ancestor-Assistant strategy can achieve the highest performance when compared to other strategies, while the discriminative naïve Bayesian classifier can achieve higher performance than the standard naïve Bayesian classifier. Furthermore, the search results are demonstrated in deep hierarchical style, which provides more informative class labels for users' browsing.

By considering that using the content rather than only the snippet can make some further improvements on performance, in the future, we will further verify the performance on the whole content of Web page instead of snippets. As mentioned in Section 6, the number of top-level categories corresponding to a query are usually less than five. We omitted the number of queries with no returned categories from the ODP for the limited coverage of human labeled Web directory. In such a situation, the method in this paper will have difficulties. It requires further research to handle such a situation. For example query expansion may be a solution to this. At present, we simply employ a flat classifier on all top-level categories to classify the search results from these queries.

9. REFERENCES

[1] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman and O. Frieder. Using manually-built web directories for automatic evaluation of known-item retrieval. *In Proc. of SIGIR, Toronto, Canada*, 2003.

[2] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. *In Proc. of CIKM*, pages 78–87, 2004.

[3] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *VLDB Journal: Very Large Data Bases*, 7(3):163–178, 1998.

[4] C.-C. Chang and C.-J. Lin. LIBSVM : a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm

[5] H. Chen and S. T. Dumais. Bringing order to the web: Automatically categorizing search results. *In Proc. of CHI'00*, pages 145–152, August 2000.

[6] P. Chirita, W. Nejdl, R. Paiu and C. Kohlshuetter Using ODP metadata to personalize search. *In Proc. of SIGIR*, Salvador, August 2005.

[7] P. Domingos and M. J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[8] S. T. Dumais and H. Chen. Hierarchical classification of web content. *In Proc. of SIGIR*, pages 256–263, August 2000.

[9] P. Ganesan, H. Garcia-Molina and J. Widom. Exploiting hierarchical domain structure to compute similarity. *Technical report, Stanford Computer Science Dept*, 2001-27, June 2001.

[10] M. Granitzer. Hierarchical text classification using methods from machine learning. *Master's Thesis, Graz University of Technology*, 2003.

[11] M. Hearst. Clustering versus faceted categories for information exploration. *Communication of the ACM*, 49(4):59–61, 2006.

[12] A. Juan and H. Ney. Reversing and smoothing the multinomial naive bayes text classifier, *In Proc. of PRIS*, Alacant (Spain), pages 200-212, 2002.

[13] B. Kules and B. Shneiderman. Categorized graphical overviews for web search results: An exploratory study using U.S. government agencies as a meaningful and stable structure. *Technical report HCIL-2004-38, CS-TR-4715, UMIACS-TR-2005-23, ISR-TR-2005-71.*

[14] F. Liu, C. Yu and W. Meng. A personalized web search by mapping user queries to categories. *In Proc. of CIKM*, pages 558–565, 2002.

[15] T.-Y. Liu, Y.-M. Yang, H. Wan, H.-J. Zeng, Z. Chen and W.-Y. Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explorations*, 7(1):36–43, 2005.

[16] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification, *In AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[17] T. M. Mitchell. Machine Learning, McGraw Hill. ISBN 0-07-042807-7. Section 6.10.

[18] D. Mladenic. Turning yahoo into an automatic web page classifier. *In Proc. of ECAI*, Brighton, UK, pages 473–474, 1998.

[19] A. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes, *In NIPS 14*, Cambridge, MA: MIT Press, 2002.

[20] B. Sergey and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[21] A. Sun and E. Lim. Hierarchical text classification and evaluation. *In Proc. of ICDM*, pages 521-528, 2001.

[22] A. S. Tanenbaum. Modern Operating Systems (the second edition) section 4.4.4, pages 217-218, New Jersey: Prentice-Hall 1997.

[23] Y. Yang. An evaluation of statistical apphroaches to text categorization. *Journal of Information Retrieval*, 1 No. 1/2:67–88, 1999.

[24] Y. Yang, J. Zhang and B. Kisiel. A scalability analysis of classifiers in text categorization. *In Proc. of SIGIR*, pages 96–103, 2003.