Deep Classifier for Large Scale Hierarchical Text Classification

Dingquan Wang, Weinan Zhang, Gui-Rong Xue, and Yong Yu

Dept. of Computer Science and Engineering, Shanghai Jiao Tong University,
Dongchuan Road. 800, 200240 Shanghai, China
{dqwang,wnzhang,grxue,yyu}@apex.sjtu.edu.cn

Abstract. In this competition, we refined a novel algorithm for classification on large scale documents with deep category structure based on a two-stage strategy known as the deep-classifier [1]. The basic idea of deep-classifier is to take advantage of the better performance of k-NN relevance search on large scale categories and the higher precision of the Naive Bayes for multi-class classification. As a result, it achieved improvement on both performance and efficiency in our experiment.

1 Introduction

In our algorithm, the training set was firstly well pruned using search method before the second stage's accurate and efficient classify. There are two issues mainly affect the performance of our classifier, the first is the large category and feature set for each document which seriously slow down the execution time, second is the deep category structure that decrees the accuracy of the result. To deal with the first problem, feature selection was carefully performed based on the vector space model of tf×idf. To deal with the second problem, we came up to a bottom up refining method by adding in the hierarchy information. Since the two stages are different evaluation aspects of the same category, the second stage is more likely to perform better after inheriting the relevant scores from the first stage, which was verified in our experiment.

2 Algorithms

2.1 Overview

In this section, we refine the deep-classification algorithm based on the one proposed by Xue. et al[1]. For a given document, the main problem of traditional approach for classification is the large scale of categories and features, which is the main difficulty to over come in this contest. Facing this problem, a two-stage approach is to extract a subset of the whole categories which is related to the given document fast, called *Search Stage*, and, sequently, the traditional explicit method of classification could be used on the extracted categories to offer the

final category as the result, called *Classification Stage*. Meanwhile, another challenge is the deep hierarchy structure of the category tree. In our algorithm, we use the hierarchy information in the Search Stage, which improves the precision of the Search Stage.

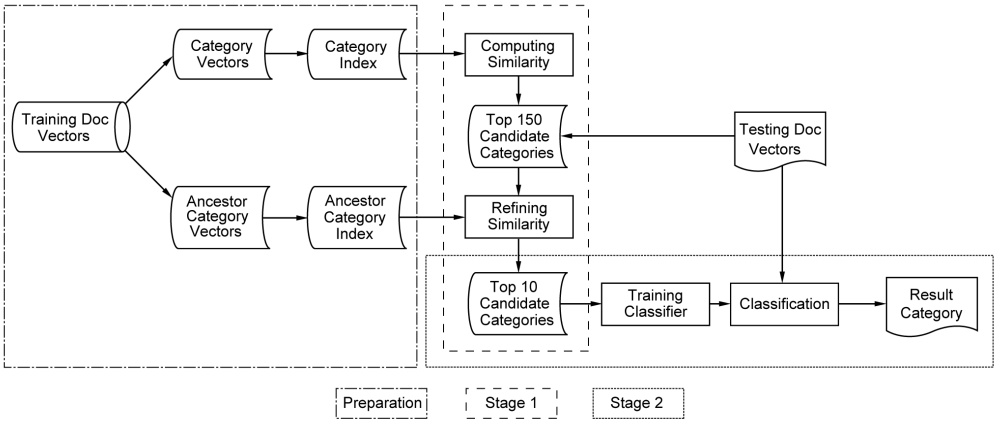

Fig. 1. Algorithm flow chart of refined deep-classification

The algorithm flow chart is shown in Fig. 1. In the *Preparation* part, we extract the category vectors by merge the document vectors belong to each category. According to the contest data, only the leaves categories of the category tree have instance of documents. Thus the ancestor categories have no document to feature them. Dealing with this case, we extract the ancestor category vectors by merge the child category vectors of them. Both the leaf vectors and the ancestor vectors are indexed in the search engine for the Search Stage.

In the Search Stage, the leaf vector index is used in the basic search part of the Search Stage, which return the top N(150, in our experiment) related categories to the given testing document; the ancestor vector index is used in the refinement part of the Search Stage, which returns the top K (10, in our experiment) related categories to the given testing document as the candidates for the Classification Stage. In our experiment, an improvement is seen after the addition ancestor information.

In the Classification Stage, the vectors of candidates are chosen to train a classifier(Naive Bayes, which works the best in our experiment). Compared with the traditional classification, this stage is based on the candidate categories from Search Stage so that for every testing case, we will train a new classifier from the candidate categories. In such case, we discover two place for improvement over the deep-classifier proposed by Xue. Firstly, for every testing case, the training data for Classification Stage is different, thus the parameters of the classifier in this stage should be different as the training data changes, which is

called *Personalized Parameter Tuning*. Secondly, as the candidate categories are recommended by the Search Stage, the ranking information of the Search Stage can be used in the Classification Stage. Both of the two improvements will be discussed in the section 2.3.

2.2 Search Stage

The main purpose of Search Stage is providing a relevant subset of category candidates for the Classification Stage. In this stage, the cosine similarity search of k-NN is chosen as the main algorithm. The input file of this stage is a set of training categories and a set of testing documents whose feature vectors were redistributed after the preparation stage and the output of this stage is top 10 categories with its relevant ranking scores. The whole stage can be separated into two parts, In the first part, a flat strategy is adopted for directly choose the top 150 leaf category candidates the its relevant ranking scores. In the second part, the ancestor information of these 150 leaf category candidates were introduced. The similarity calculation is secondly performed, but this time we calculating the similarity between the test and the candidate categories' ancestor. The final score of the these 150 candidates is a combination of scores from the first part and the second part. After seizing the final search score, an efficient top-K search is processed for extracting 10 most similar categories from 150 categories as the relevant subset of category candidates for the Classification Stage.

2.3 Classification Stage

Given the testing document and then the candidates categories and with the help of their relevance score in Search Stage, the Classification Stage builds up a traditional classifier and classify the testing document. Specifically, in our experiment, we get the best performance and efficiency using Naive Bayes Classifier to classify the 10 candidate categories.

Personalized Parameter Tuning In the standard Naive Bayes Classifier based on the multinomial model and Laplacian smoothing yields, the maximum likelihood estimation of a word ω_i and class c is:

$$p(\omega_i|c) = \frac{N_c^i + \alpha}{N_c + V\alpha} \tag{1}$$

where N_c^i stands for the number of times that ω_i occurs in the documents of c and N_c stands for the total word number of the documents of c. The total vocabulary size of the training documents is noted as V and α is the parameter of this model. In the deep-classifier, for each testing case, the Search Stage will offer a set of candidate category, of which the total vocabulary size V changes case by case. In the standard NB model, the parameter α is tuned against V to refine the performance. Therefore, in deep-classifier, as V changes against every testing case, the parameter m should also changes. We call the tuning of

the parameters of the classifier against the testing case *Personalized Parameter Tuning*.

We divide the vocabulary size in some continuous regions after the analysis of the vocabulary size distribution. For each region, the parameter α has a different value(in NB model, the larger V is, the smaller α is). If α unchanges in different regions, the scenario comes back to basic deep-classifier.

Two Stage Score Combining The K-size candidate category set from the Search Stage is not just a set in fact. As is discussed in the section 2.2, the Search Stage will return the candidate categories with the relevant score to the testing document. These scores suggest the possibility of each candidate category as the final result. After analysis of the relevant score with the final result, we discovered that the higher the score(rank) of a candidate category is, the more likely it is to be the right answer. In such case, the relevant score from the Search Stage can refine the performance of the Classification Stage.

We record the scores given by both stage and merge the two score to the final score of a candidate category. For a class c, the Equation is as below:

$$s_c = s_{1,c}^e \cdot s_{2,c} \tag{2}$$

where $s_{1,c}$ and $s_{2,c}$ stand for the score from the Search and Classification Stage respectively. The parameter e (< 1) limits the importance of the score from in the Search Stage.

Feature Selection In deep-classifier, for every test case we should train a new classifier, which takes much time. Feature selection on the candidate document feature set is significantly useful to cut down the time of the Classification Stage with little reduce of the time used in this stage.

3 Experiment and Evaluation

3.1 Experiment Specification

In this section, we describe the details of our experiment.

Category Vector Building We merge the document vectors by plus the value of the same features and combination of all the features. To get the ancestor vectors, we merge the child categories to their ancestor with the same method of merging document vectors. This process is simple enough but so useful that until now we can still not find a better method.

K-NN Similarity According to the Lucene source code [2], the tf×idf vector is not truly the value of $tf \cdot idf$. Instead, the value is $tf^{0.5} \cdot idf$. The exponent

parameter can tunes for different data set. In our experiment, we use $tf^{0.625} \cdot idf$ as the value of each vector feature.

The first part of Search Stage, we extract $N = 150$ candidates with a relevant score for each one. In the second part of Search Stage, we calculate the relevance score of the father categories of the N candidates. Then we combine the relevant score of each candidate category and their father. The final relevant score of class c in Search Stage is

$$rsc = s_c \cdot s_{f(c)}^{0.52} \tag{3}$$

where s_c stands for the first part score for c and $s_{f(c)}$ stands for the second part score for the father category of c.

Feature Selection We tried several approaches for feature selection, such as CHI-χ^2, IG, tf×idf. In our experiment, tf×idf performance the best that it doubles the speed of training with a reduce of 1% of performance on the local test data(validation.txt). However, Search Stage takes almost 12 times of time to the Classification Stage, thus the feature selection cannot improve the efficient of the whole work. In our submit code, we do not include the feature selection procession in order to make sure the best performance.

Personalized Parameter Tuning Without the feature selection, the vocabulary size for each test case is significantly different. In the test of local large data, the distribution of vocabulary size and the best value of α is given in the Table below.

vocabulary	distribution rate	tuned α
1∼2000	2.95%	0.36
2001∼3500	19.1%	0.27
3501∼6000	38.0%	0.16
6001∼10000	18.4%	0.14
10001∼ ∞	21.5%	0.08

Two Stage Score Combining Similar to the combination between the two parts of Search Stage, for class c the combination $cs_c$of two stage scores in given below:

$$cs_c = rs_c^{0.12} \cdot ns_c \tag{4}$$

where rs_c stands for the relevant score from Search Stage and ns_c stands for the score from Naive Bayes Classifier in Classification Stage. The exponent parameter can also be tuned against the data.

3.2 Experiment Conditions

Computer Hardware AMD athlon(tm) 64*2 Dual, Core Processor 5000+, 2.61GHz 7.75GB of RAM.

Operation System Microsoft Windows Server 2003 R2, Enterprise x64 Edition, Service Pack2.

3.3 Results

In section we mainly list the performance and the efficiency of the refined deep-classification on the dry-run data set.

TASK	Test Case	Hit Case	Precision	Time Used	Time per Case
Basic	1858	919	0.4946	146s	0.079s
Cheap	1858	829	0.4462	119s	0.064s
Expensive	1858	925	0.4978	158s	0.085s
Full	1858	983	0.5291	167s	0.090s

3.4 Computational Complexity Analysis

Suppose the training set have n documents covering c categories the testing set have m documents, each document have k features on an average. The complexity of the Prepare Stage is $O(nk)$. The complexity of the Search Stage is $O(cmk)+O(c) = O(cmk)$. The complexity of the Classification Stage is $O(cmk)$. So the over-all complexity is $O(cmk)$.

4 Conclusion and Future Work

In this paper, we refined the deep-classifier algorithm in both stages and get improvements on both performance and efficiency. In the future work, some parts of the algorithm will be research more deeply such as the category vector extraction from the document vector, more utilization of hierarchy information in the Classification Stage. If we can obtain the sequence of the words in the document, the N-gram method could be used, which improved the performance significantly as described in the Xue's paper [1].

References

1. Xue, G.R., Xing, D., Yang, Q., Yong, Y.: Deep Classification in Large-scale Text Hierarchies. In: Proc. of SIGIR'08, pp. 619–626. Singapore(2008)
2. Lucene Apache Website, `http://lucene.apache.org`