A two stage classification approach for large scale heirarchical classification

Vaijanath N Rao and Rohini Uppuluri

AOL Online India Pvt. Ltd, Bangalore, India,
vaijanath.rao@teamaol.com,rohini.uppuluri@teamaol.com

Abstract. This paper describes our approach to the second pascal large scale hierarchical text classification challenge. In the training phase, we transform the documents into a reduced dimension using random projection and learn a classification model consisting of a document model and a class model. In the testing phase, we follow a two staged approach. In the first stage, we identify candidate classes among the huge set of classes available and in the second phase, we use classification model to compute the final class label prediction. We present our experimental results on the challenge data sets.

1 Introduction

We participated in the first task in second pascal challenge on large scale hierarchical text classification and we describe our approach and the results in this paper. Our approach consists of two phases: the first is the training phase in which we learn a classification model and the second is the testing phase where we test our classification model against a given document. In the rest of the paper, we describe both the phases in detail.

2 Training

In the task, each document consists of feature identifiers and feature weights along with class labels. Each feature of document is considered as a dimension when representing the document in vector space model. In the training phase, we first reduce the dimensionality of documents and compute a compact representation of document. We then learn the classification model on the compact document in the reduced dimension. We have experimented with different methods for learning the classification model which we describe in the later part of this section.

2.1 Dimensionality Reduction

Reducing dimensions of a document to only include the key features also called the *latent semantic dimensions* has gained increased attention in the recent years. There have been many approaches to dimensionality reduction including probabilistic latent semantic analysis (PLSA) [6], principal component analysis (PCA), singular value decomposition(SVD)

[10] and other matrix factorization methods. Most of these methods are computationally expensive and not incremental. In order to scale with the huge amount of data we are dealing with, a computationally simple method that can be computed in an incremental fashion yet computes meaningful reduced dimensions is desirable.

Random Projection

Random projection [1, 7] is an approach which computes the reduced dimensions by chosing them at random which are nearly orthogonal. The reduced dimensions computed using random projection are comparable to an orthogonalization method like SVD though it is not so computationally expensive(see [9]).

In random projection, the original d-dimensional data is projected to a k-dimensional ($k \ll d$) subspace through the origin, using a random kXd matrix R whose columns have unit lengths. Using matrix notation where X_{dXN} is the original set of N d-dimensional obervations, the projection of the data onto a lower k-dimensional subspace is as showm in Eq (1).

$$X^{RP}_{kXN} = R_{kXd}X_{dXN} \tag{1}$$

Semantic vectors package[3] is an open source implementation of random projection. We have implemented the same algorithm in MapReduce to handle the scale we are dealing with.

Implementation of Random Projection using Map-Reduce

One of the ways to handle large scale data is to distribute computations across multiple machines. MapReduce[2] is a programming model and an associated implementation for processing large data sets in a distributed manner. In the MapReduce model, computations are split as a sequence of map and reduce steps. The map step consumes a stream of key and value tuples and outputs a set of (possibly different or amended) key and value tuples. In the reduce step, all tuples with same key are brought and processed together. Hadoop[4], an apache project is an open source implementation of the MapReduce Programming Model and HDFS, an open source distributed file system that provides the underlying storage. We have used Hadoop and HDFS in our work.

Our implementation of random projection of the data consists of three MapReduce jobs. The first job takes the training documents and assigns a unique id to reach document and also generates a random vector of the document in the specified dimension. The unique id is generated using the hash of combination of all feature ids present in that document. In the second job, we compute feature random vector and in the third job, we compute the final document and term random vectors. Details of the map reduce jobs can be found in Table 2.1.

2.2 Learning the classification model

Our classification model consists of *document model* and *class model*. We need a document model for compact representation of all the documents in the training data with easy to search capability. In addition,

Job1 - Document Random Vector Computer		
	Input(key,value)	*Output(key,value)*
M	(Line No., doc Contents)	(doc Hash, doc Contents)
R	(doc Hash, doc Contents)	(featureId , <doc rand vec,doc Contents>)
Job2 - Feature Random Vector Computer		
M	Job1 Output	(feature Id, <norm feature vec, doc Contents>)
R	mapper output	(feature Id, norm feature vec)
		(doc Hash, <norm feature vec, wt, classes>)
Job3 - Final Dimensionality Reducer		
M	second job output	(feature Id, norm feature vec)
		(doc Hash, <norm feature vec, wt>)
		(class, <norm feature vec, wt>)
		(doc Hash, class)
R	mapper output	(feature Id, norm feature vec)
		(doc Hash, <norm doc vec,classes>)
		(class, norm class vec)

Table 1. Details of Map-Reduce implementation of Random Projection: M- Mapper, R-Reducer of the mapreduce job

we wanted the approach to scale and be incremental. We need a class model to capture the class representation and the capability of finding a class given a document.

We use Lucene[8], an open source search engine to build our document model. The document model is built on the output of the map reduce jobs and consists of three types of lucene documents. The first type is class document consisting of the class information and the corresponding normalized class vector and the second type consists of document and normalized document vector and the third type is feature document consisting of the feature and the corresponding normalized feature vector.

The second model we learn is the class model. In learning the class model, we experimented with two approaches. In the first approach, we used a simple counts based approach called *class prototypes classifier*, in the second approach, we used *feature weighting classifier*.

In class prototypes classifier(CPC), we represent each class as a vector of feature and the corresponding weights. The class prototype for class C_i where $C_i = (W_{f_1}, \ldots W_{f_k} \ldots W_{f_n})$ and

$$W_{fn} = \frac{\#(C_i, f_n)}{\sum_{C_i} \#(C_i, f_n)} \tag{2}$$

where $\#(C_i, f_n)$ is the number of times the feature f_n appears in class C_i.

Feature weighting classifier(FWC) is a multi class classification algorithm proposed by Hassan et al[5]. It has shown to have outperformed naive bayes and comparable to SVM. We chose to experiment this approach due to its simplicity yet effectiveness. In the training phase, it first computes feature frequencies per class in one pass over the training data set. Then the information gain and feature support in each class is computed.

Let a_{ij} be the number of documents of class c_j where feature f_i occurs, let b_j be the number of documents in class c_j. The support of feature f_i in class c_j is computed as as

$$p_{ij} = \frac{a_{ij}}{b_j}$$

The information gain of feature f_i is computed as

$$IG = \sum_{cinC} \sum_{x_i in 0,1} P(c, x_i) \log_2 \frac{P(c, x_i)}{P(c)P(x_i)}$$

A linear combination of the support of a feature and information gain is the final score of each feature class pair. During the testing phase, a classification score is computed which is a dot product of the document vector the class weights vector. We encourage the reader to read the paper by Hassan et. al [5] for further details.

3 Classification

Given a document, we follow a two stage approach for classifying it. In the first stage we compute candidate classes or categories and in the second stage, we compute the actual class prediction. This two stage approach helps us in narrowing the candidate categories to only a few and after which we can apply any classifier consider only the candidate categories and picking the best class (often classes).

3.1 Computing Candidate Classes

Given a document, we compute K nearest neighbour documents from the document model using cosine similarity. We then consider the class label for each of these documents and compute the *candidate class score*. Let d_i be the document we need to classify and S_{C_i,d_i} represent the candidate class score for class C_i and K represents the neighbours of the document d_i. S_{C_i,d_i} is computed as shown in Eq (3).

$$S_{C_i,d_i} = \sum_{x \in K} cosine(d_i, x) \tag{3}$$

We sort the candidate class scores in descending order and pick the top N classes.

3.2 Computing Class Predictions

We apply the learned class model as described in Section 2.2 on the top N classes to compute *category score* for each of the classes. We then combine the class candidate score and the category score of each class to get the *class label score* and sort them in descending order and pick the top class label(s).

Let d_i be the document and C_i be a class and Cl_{C_i} denote the class label score and Cat_{C_i} denote the category score. Then, the class label score is computed as shown in Eq (4) and $g(x)$ is computed as shown in Eq (5). δ is a smoothing parameter which we have set to 0.001 in our work based on experimentation.

$$Cl_{C_i} = \alpha \; S_{C_i,d_i} + \beta \; g(Cat_{C_i}) \tag{4}$$

$$g(x) = \begin{cases} x & \text{if } x > 0 \\ \delta & \text{otherwise} \end{cases} \tag{5}$$

We pick the class label C_i with the highest class label score Cl_{C_i} as the class for the given document. All the classes within a given threshold(set at 95% of the top class label score) are chosen as additional classes for the given document. Incase, the top class label score is equal to 1, then the threshold is changed to 100%.

4 Evaluation

In this section, we describe some details about the experiments we have performed. We first describe the hardware details and the training and test times and then we describe the actual evaluation results in task 1 of the challenge.
Our System is implemented in Java. It runs on a hadoop cluster of 6 machines (PCs) with each machine being a Intel Core Duo 2Ghz CPU with 4 GB memory. We have used the cluster set up for parallel and distributed execution of both training and testing. The approximate training and classification times are given in Table 2.

Task Name	**Training Time**	**Testing Time**
Dimensionality Reduction		
Random Vector	5 mins	1 hour
Classification		
CPC	2 mins	15 mins
FWC	10 mins	20 mins

Table 2. Details of the training and testing times

We performed experiments using our system in Task 1 of the challenge. For dimensionality reduction, we have set the reduced dimension to 500 (dimensions per vector) in our work based on experimentation. We have used FWC classifer for the results presented in this section. The results are summarzied in Table 3.

5 Conclusions and Future Work

In the above experiments we have shown that our approach works well when we treat the categories as flat hierarchy. The two stage approach

Accuracy	0.262414
MGIE	3.47579

Measure	**F-Measure**	**Precision**	**Recall**
EB	0.263099	0.263491	0.263414
LBMa	0.129597	0.246935	0.146256
LBMi	0.261925	0.261246	0.262606

Table 3. Results on the challenge data sets. EB - Example Based, LBMa - Label Based Macro, LBMi - Label Based Micro, MGIE - Multi-label Graph Induced Error

works very well as long as the first stage can reduce noise in terms of most probable categories and our implementation of random projection ensures we do get the best possible near neighbours.
We believe that the hierarchial categorical information is more informative than a flat hierarchy. In future, we are planning to explore the use of parent or hierarchy information to study how it effects our approach. We are also working towards reducing the testing time of the dimensionality reduction and also explore other methods for the same.

References

1. Bingham, E., Mannila, H.: Random projection in dimensionality reduction : applications to image and text data. In: KDD 01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 245–250. ACM, New York, NY, USA (2001)
2. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: Communications ACM. vol. 51(1), pp. 107–113 (2008)
3. Dominic Widdows, K.F.: Semantic vectors: a scalable open source package and online technology management application. In: LREC (2008)
4. Hadoop. open source distributed framework, available at: http://hadoop.apache.org.
5. Hassan H Malik, Dmitriy Fradkin, F.M.: Single pass text classification by direct feature weighting. In: Knowledge and Information System (2010)
6. Hofmann, T.: Probabilistic latent semantic analysis. In: Uncertainty in Artificial Intelligence (UAI99). Stockholm, Sweden (1997)
7. Kanerva, P.: Sparse Distributed Memory. MIT Press (1988)
8. Apache lucene, an open source search engine, http://lucene.apache.org
9. Sahlgren, M.: An introduction to random indexing. In: Proceedings of the Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering (TKE). Copenhagen, Denmark
10. Trefethen, L.N., Bau, D.: Numerical Linear Algebra. S.I.A.M (1997)