Large Scale Multi-label Text Classification of a Hierarchial DataSet

Animesh R[1], Dhiresh Jain[1], Arvind Hudli[1] and Darshan Dorai[1]

[1]Department of Computer Science, M S Ramaiah Institute of Technology, Bangalore

***Abstract*—Hierarchical data is becoming increasingly prominent, especially on the web. Wikipedia is one such example where there are millions of documents that are classified into multiple classes in a hierarchical fashion. This gives rise to an interesting problem of automating the classification of new documents. As the size of the dataset grows, so does the number of classes. Further, there seems to be sparsity issue even with the increase in the dataset. Therefore, this poses a challenge to classify data in such a manner. We present two different algorithms based on text categorization : Rocchio algorithm and kNN. We implement and compare the above mentioned methods to better understand the approach to take in classifying hierarchical data.**

I. Introduction

Text categorization or classification, is the task of categorizing natural language texts with theme based classifications from a predefined set. TC has been widely used in several real-world applications, like spam filtering, organization of large scale web pages and online news classification. Many classification algorithms have been used for text classification, like decision trees, support vector machines (SVMs), k-nearest neighbors(kNN s) , etc.

Among them, centroid-based classifier (CC) is noteworthy for its high efficiency and robust nature. Generally, the computational complexity of CC while training is roughly proportional to the total number of documents and terms in the training set, which is especially interesting for large-scale text classification tasks. Also, CC matches a new document to dissimilar centroids in classification, which allows it to dynamically calibrate for classes with dissimilar densities. The basic idea of CC is to use all the training records belonging to one category to build centroid vectors, and finally allocate a new document to the category with the most similar centroid. Therefore, the computational complexity while classification is proportional to the number of classes that exist at that time. However, good metrics should be used to compute the centroids for better accuracy.

k-Nearest Neighbour(kNN) is a kind of lazy learning where no training is required. It does not attempt to generalize the training data set and delays the computation until a new document arrives. Though the computational complexity of this algorithm during classification is proportional to the size of the training set, it is more expressive than CC and can handle complex classes with relative ease.

Many user-centric and content-driven web applications have been flourishing over the past few years. A few examples of these applications would include blogs, wikis and resource sharing systems. The need to organize such haphazard content into categorized resources has driven the motivation for text categorization.

There also exists a social aspect when it comes to user-driven textual tagging. Multiple users are able to freely tag several documents in real time. But the simplicity of this approach certainly comes at the expense of several drawbacks. Firstly, the tags chosen by the users are highly dependent on their personal opinions and their preferences. Moreover, people might be describing the same entity based on different granularity. This process leads to the generation of noisy tags and makes it extremely difficult to extract the relevant labels. Secondly, users might also use polysemous words (words with different but related senses) to tag the textual web resource. The absence of semantic contrast in tags might eventually lead to unsuitable connection between items. Semantically similar

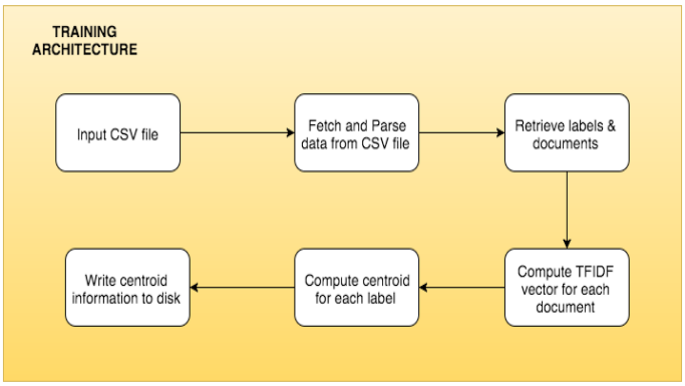

Fig. 1. Training architecture used in this paper.

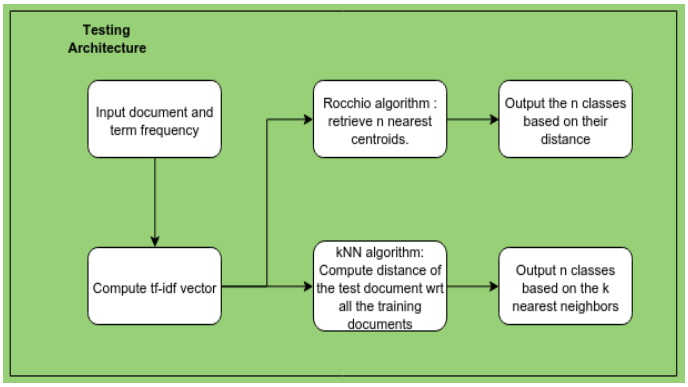

Fig. 2. Testing architecture used in the paper.

tags also drastically increase the redundancy in data, leading to reduced information recall. Finally, users tend to assign a very small number of labels to an object.

Text classification is used to overcome these problems. It automatically suggests a concrete set of suitable tags for the textual web resources. Some of the related work in the field includes graph/text based approaches in conjunction with collaborative filtering. In this paper we model the existing problem as multilabel text classification task.

II. Multilabel Classification

Traditional single-label classification focuses on learning from a collection of examples that are associated with a single label λ from a set of disjoint labels L, $|L| > 1$.The learning task is known as binary classification when $|L| = 2$. On the other hand, it can also be known as filtering in the case of web and textual data. However, multi-class classification is when $|L| > 2$. The set of labels $Y \in L$ is generally associated with the examples in multilabel classification.

Multilabel classification is a demanding research problem that can be witnessed in several modern day applications such as music categorization, semantic classification of images and protein function classification. In the past, multilabel classification has mainly attracted the attention of researchers working on text categorization, as each member of a document collection is generally a part of more than one semantic group.

Multilabel classification methods can be classified into two different groups: i) problem transformation, and
ii) algorithm adaptation. The first class of methods are algorithm independent. They modify the multilabel classification task into many single-label classifications, regression or label ranking tasks. The next group of methods extend specific learning algorithms in order to manage multilabel data directly. The most commonly-used problem transformation method, called Binary Relevance (BR Learning), takes the prediction of each label as a separate binary classification task. One binary classifier is learnt $h : X \rightarrow \{\neg\lambda, \lambda\}$ for each different label $\lambda \in L$. It transforms the original data set into $|L|$ datasets D that contain all examples of the original data set, marked as though the labels of the original example had λ and as $\neg\lambda$ otherwise. It is the same solution used in order to deal with a multiclass problem using a binary classifier, commonly known to as one-against-all or one-versus-rest.

III. Curse of Dimensionality

Curse of dimensionality is a phenomenon that refers to the fact that some problems are very complex to compute because of the large quantity of features. This means available solutions often overshoot available computing resources. The original meaning was published in [Bellman, 1961] and this phenomenon in the information retrieval domain has been first mentioned in [Koller and Sahami, 1997]. The curse of dimensionality also leads to several problems closely associated with big data. In high dimensional vector spaces, data is very sparse and in order to estimate any arbitrary parameter, one must have several samples to achieve an admissible level of accuracy. The dimensionality of the vector space increases approximately greater than exponentially with the number of

samples. Moreover, this acutely restricts possible applications because the resulting computer power demand is very high and greatly restricts a potential set of solutions. Of course, this completely applies to the area of document classification, as is mentioned in [Zervas, 1999].

In real world document classification problems, the amount of data collected for training purposed would always be less. This describes that the rising number of data vectors, is proportional to the number of features. This occurs due to the procedure used to create document vectors. Therefore, it is highly likely that by adding new documents into a set, one gets supplementary terms and so the number of features increases. One solution would be to take all feasible terms of a language and use these terms as features from the beginning. Assuming one would be able to do that, there would be at least two new issues one would have to overcome:

1. Assuming approx. one hundred thousand terms/features in order to get reasonable estimates, we would need to have more than or at least the same number of documents. This is not always possible. 2. If no terms occur in any document, the value of feature for that document is considered zero. The variance is also considered to be zero. This feature would not add any additional information for classification. Therefore, it is clear that in the area of document classification, it is highly likely that we will face a situation when there are more features than observations. It must be noted that some studies have revealed that some classifiers may give a low generalisation error even in cases when the total number of data vectors is lower than the number of features. This is called peaking phenomenon.

IV. Design

The training dataset is stored in a csv file. We parse the data and store the individual elements such as document id, term frequency, inverse document frequency, centroids and td-idf in a database. All the intensive training and prediction/classification is done using this data.

We use parallel programming with the help of CUDA(Compute Unified Device Architecture) platform for the data intensive operations of finding distances between

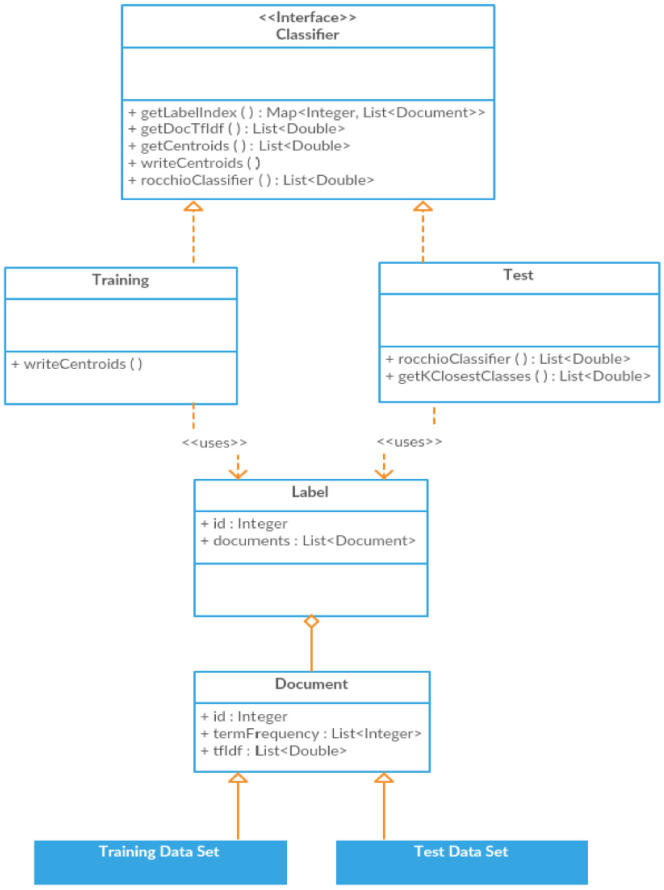

Fig. 3. Class Diagram for the proposed architecture

any two vectors of large dimensions. Since the prediction algorithm has no inter-dependencies nor does it have to follow any particular order, we can successfully employ parallel programming without compromising on the accuracy or correctness of the prediction had we done it serially.

CUDA is a platform and an API developed by NVIDIA that allows users to use Graphic Processing Unit for general computing purposes. GPU programming is much faster than its CPU counterpart, and provides efficient parallelism.

For each of the test document, we compute its vector form. We run the prediction/classification algorithm on the GPU. Since there are millions of documents in the training set, a considerable amount of time is saved in the prediction phase without any compromise on the output quality. This mitigates a major disadvantage that each of the algorithm face, that of the intensive operations that they have to perform during prediction phase. However, this problem will still exist

depending upon the size of the training set. The efficiency of the algorithm would depend on the accuracy required.

V. Dataset

The dataset used in this paper was obtained from the Kaggle challenge titled 'Large Scale Hierarchical Text Classification'. The challenge uses Wikipedia to generate a dataset composed of an enormous number of documents. The dataset is multi-label, hierarchical, and multi-class. There are roughly 3,25,000 classes and approximately 24,00,000 documents. The challenge is an extension of successful challenges on large-scale hierarchical text classification.

The hierarchy file consists of a representation the hierarchy of classes. Each line establishes a relationship between a parent node and a child node. For example, the line:

```
347 73
```

should be read as node 347 is parent of node 73.

The format of each data file uses the libSVM format. Each line corresponds to a sparse document vector and is organised as:

```
label, label, label ... feat:value ...
feat:value
```

Label is an integer and can be mapped to the group to which the document vector belongs. Every such vector may belong to more than one category. The pair feat:value corresponds to a feature greater than zero. Feat represents a term and value corresponds to the weight of the term in the document.

VI. Implementation

In machine learning, the nearest centroid classifier or nearest prototype classifier is a model that divides the vector space into regions centered on centroids, one for each of the classes. The centroid is computed based on the tf-idf metrics aggregated from all documents belonging to the same class.

When used in text categorization, the nearest centroid classifier is also known as the Rocchio classifier because of how similar it is to the Rocchio algorithm for relevance feedback. An elongated version of the nearest centroid classifier has found many uses in the medical domain, specifically classification of tumors.

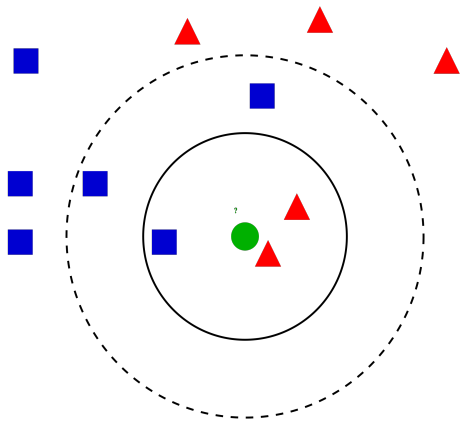

Fig. 4. Example of k-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

Algorithm 1 Rocchio's algorithm

procedure TRAINROCCHIO (C, D)
  for each $c_j \in C$
  do $D_j \leftarrow \{d : (d, c_j) \in D\}$
  $\mu_j \leftarrow$ mean of tfidf vector
  return $\{\mu_1, ..., \mu_j\}$

We first parse and store the training data in a database. We then compute the idf values for each of the terms. The tf-idf calculation follows. Centroids for each label is then computed by considering the average of all its corresponding document TFDIF vectors. These centroids would then be used to determine the most accurate label of any given test document. Centroids near the document vector in the vector space can be considered to be one of the tags for that document.

For the classification of a test document, we use parallel programming. We run the prediction part of the algorithm on

Algorithm 2 kNN algorithm

procedure KNEARESTNEIGHBORS(k, D, T)
  distances = []
  for each $d_j \in D$
  $distances.append(distance(d_j, T))$
  sort $distances$
  return $\{distances_1, ..., distances_k\}$

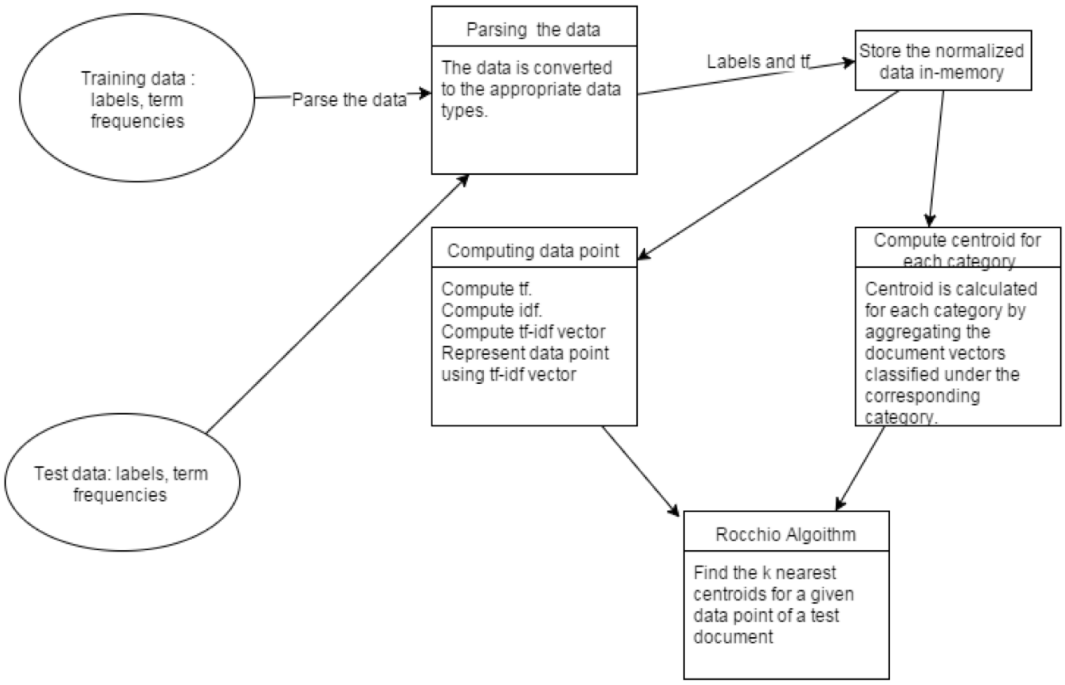

Fig. 5. Data flow diagram of the proposed architecture.

a GPU, using the CUDA platform. We compute the distances of each of the centroid with respect to the test vector parallely. Since the computation of the distance vectors are independent of each other, there is no compromise on the accuracy of the prediction in classifying the documents. We keep track of only n centroids that are nearest to the test vector.

In the case of kNN, we first fix the value of the parameter k. We then compute the distance between each of the training document vector and the test document vector. Since the training data is huge, we compute these distances parallely. We then determine the k nearest neighbors. The n nearest classes are determined based on a score calculated as the number of these k documents that belong to a class. Hence, more the number of documents belonging to a class, more is its score.

VII. Testing and Comparison

In this paper, we compare two text classification algorithms - Rocchio's and kNN. In kNN, we fix the 'k' initially to 5. We then compute the distance of the new document vector with the vectors of all other documents by the formula :

$$d = \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2 ...} \tag{1}$$

where a, b, etc. refer to the feature values in the corresponding vectors. This fundamentally means that in order to classify a given new document, we must compute its distances with the TFIDF vector of all the other documents. This concept is obviously flawed and is very inefficient. kNN algorithm does not scale well with large data and higher dimensionality. This is where Rocchio's algorithm is beneficial. Instead of iterating through all the documents, we iterate through all the categories in the dataset. As stated above, each category is represented by a centroid. In the prediction phase of this scheme, we compare the centroids of the categories and the TFIDF vector of the new document. Since there are numerous mathematical operations involved in computing the distances between two high dimensional vectors, we make use of the CUDA enabled GPU to compute and compare the distances.

VIII. Evaluation & Results

The evaluation metric for the Kaggle Competition is as follows:

$$MaF = \frac{2 * MaP * MaR}{MaP + MaR} \tag{2}$$

For a set of classes $C\{c_1, , c_k\}$ macro precision and macro recall are calculated as follows:

$$MaP = \frac{\sum_{i=1}^{|C|} \frac{tp_{c_i}}{tp_{c_i}+fp_{c_i}}}{|C|} \quad (3)$$

$$MaR = \frac{\sum_{i=1}^{|C|} \frac{tp_{c_i}}{tp_{c_i}+fn_{c_i}}}{|C|} \quad (4)$$

where $tp_{c_i}, fp_{c_i}, fn_{c_i}$ are the true positives, false positives and false negatives respectively for class c_i.

The score received by this framework was 0.28931 for the Rocchio based approach and 0.23088 for the kNN approach. As it was elaborated earlier, kNN is computationally more intensive as compared to Rocchio's algorithm. The Rocchio based approach was ranked 5 of 150 on the Kaggle leaderboard.

IX. Scope and Future Work

This work evaluated Rocchio's algorithm and the kNN algorithm to support multilabel text classification on the Kaggle LSHTC dataset. The extreme computing power from the Nvidia GPU has been exploited in order to compute and compare the distances between individual TFIDF vectors and the collection of clusters during the prediction process. The TFIDF metric does not assign a score to documents where the key term does not appear in, regardless how informative a particular term may be. Moreover, TFIDF does not pair words with their corresponding synonyms. The main advantage of TFIDF is that it is easy to compute and can be easily parallelized.

Future work includes more research on unsupervised learning in multilabel text classification. Particularly on the Internet, we would not have the comfort of having structured data. Further, methods which involve complex syntax and semantics have proven to be less accurate as compared to the naive approaches. Hence more research could be on improving the existing algorithms which utilize the syntax and semantics of text. Which features work well together, and which are redundant? Are different features better for different corpuses? Are different classifiers better for different features? The solutions to these questions could be a part of the future work.

X. Conclusion

We have proposed a vector space based architecture to predict the labels of a given document. We have then compared Rocchio's and kNN algorithm and elaborated why Rocchio's algorithm is more efficient and accurate than kNN. Regardless of the primitive nature of these methods, its drawbacks are negligible considering the massive scale of the dataset. The computational potential in the CUDA based Nvidia GPUs have also been exploited in order to compute the distances between the high dimensional vectors.

References

[1] Kaggle LSHTC Challenge, http://kaggle.com/c/lshtc

[2] Krzysztof Sopya, Pawe Drozda, Przemysaw Grecki. SVM with CUDA Accelerated Kernels for Big Sparse Problems. 2012

[3] V. Vaitheeshwaran, Kapil Kumar Nagwanshi, T. V. Rao. Multicore Processing for Classification and Clustering Algorithms, 2012.

[4] Basu, T. Effective Text Classification by a Supervised Feature Selection Approach 2012

[5] A Sun et. al. Short text classification using very few words. 2012

[6] CH Wan, LH Lee, R Rajkumar, D Isa - Expert Systems with Applications. A hybrid text classification approach with low dependency on parameter by integrating K-nearest neighbor and support vector machine. 2012

[7] YS Lin, JY Jiang, SJ Lee - Knowledge and Data Engineering. A similarity measure for text classification and clustering. 2014

[8] AK Uysal, S Gunal - Knowledge-Based Systems. A novel probabilistic feature selection method for text classification. 2012

[9] JY Yoo, D Yang. Classification Scheme of Unstructured Text Document using TF-IDF and Naive Bayes Classifier. 2015

[10] M Ghiassi, M Olschimke, B Moon, P Arnaudo. Automated text classification using a dynamic artificial neural network model. 2012

[11] P. McCullagh, J. Yang, How many clusters? Bayesian Analysis 3 (2008) 101120.

[12] J Read, B Pfahringer, G Holmes, E Frank. Classifier chains for multi-label classification. 2011

[13] C Bielza, G Li, P Larranaga. Multi-dimensional classification with Bayesian networks. 2011

[14] J Read, A Bifet, G Holmes, B Pfahringer. Scalable and efficient multi-label classification for evolving data streams. 2012

[15] L Zhou, Z Yu, J Lin, S Zhu. Acceleration of Naive-Bayes algorithm on multicore processor for massive text classification. 2014

[16] S Canuto, T Salles, MA Gonalves. On Efficient Meta-Level Features for Effective Text Classification. 2014