**M S Ramaiah Institute of Technology**

(An Autonomous Institute, Affiliated to VTU)

MSR Nagar, MSRIT post, Bangalore-54

A Dissertation Report on

Large Scale Classification of Multilabel Documents

Submitted by

Animesh Ramesh 1MS12CS009

Arvind Hudli 1MS12CS014

Darshan Dorai 1MS12CS022

Dhiresh Jain 1MS12CS028

*in partial fulfillment for the award of the degree of*

# *Bachelor of Engineering in Computer Science & Engineering*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**M.S.RAMAIAH INSTITUTE OF TECHNOLOGY**

**(Autonomous Institute, Affiliated to VTU)**

**BANGALORE-560054**

[www.msrit.edu](http://www.msrit.edu), **May 2015**

# 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
* kNN

We implement and compare the above mentioned methods to understand better the approach to take in classifying hierarchical data.

Contents

1. INTRODUCTION
   1. General Introduction
   2. Statement of the Problem
   3. Objectives of the project
   4. Project deliverables
   5. Current Scope
   6. Future Scope
2. PROJECT ORGANIZATION
   1. Software Process Models
   2. Roles and Responsibilities
3. LITERATURE SURVEY

3.1 Introduction

3.2 Main Body

3.3 Conclusion of Survey

1. SOFTWARE REQUIREMENT SPECIFICATIONS

4.1 Product Overview

4.2 External Interface Requirements

4.2.1 User Interfaces

4.2.2 Hardware Interfaces

4.2.3 Software Interfaces

4.2.4 Communication Interfaces

4.3 Functional Requirements

4.4 Software System Attributes

4.4.1 Reliability

4.4.2 Availability

4.4.3 Security

4.4.4 Portability

4.4.5 Maintainability

4.4.6 Performance

4.5 Performance Requirements

4.6 Database Requirement

4.7 Design Constraints

4.8 Other Requirements

1. DESIGN
   1. Introduction
   2. Architecture Design
   3. Graphical User Interface
   4. Class Diagram and Classes (represent Inheritance, Aggregation and Association)
   5. Sequence Diagram
   6. Data flow diagram
   7. Metric calculation
2. IMPLEMENTATION
   1. Tools Introduction
   2. Technology Introduction
   3. Overall view of the project in terms of implementation
   4. Explanation of Algorithm and how it is been implemented
   5. Information about the implementation of Modules
3. TESTING

7.1 Results and Snapshots

1. CONCLUSION & SCOPE FOR FUTURE WORK
2. REFERENCES

INTRODUCTION

Text categorization or classification, is the task of categorizing natural language texts with theme based classifications from a predefined set. It 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.

**Problem Statement**

Given a very large dataset of Wikipedia documents, classify each document into one or more categories.

**Project Objectives**

* Given a document and its term frequencies, we need to predict the labels to which the document belongs.
* To minimize the computation load on the client machine.
* To improve the prediction accuracies of the prediction model.
* To reduce the time taken for prediction.

**Project Deliverables**

* The output of the model is in plain text format.
* Each line of the file contains the predicted categories of the hierarchy chosen by the model for the corresponding vector of the test file.
* Only leaves of the hierarchy are valid classification answers.

Id Predicted

1 23, 4567, 23333

2 542, 11

3 986, 872, 98293

**Current Scope**

* Given a new document, our model would automatically be able to predict the categories to which it belongs.
* Very helpful for automatic tagging of documents on Wikipedia or Hierarchical Collection of Documents.
* Easier to navigate through hierarchical data as compared to unstructured data.

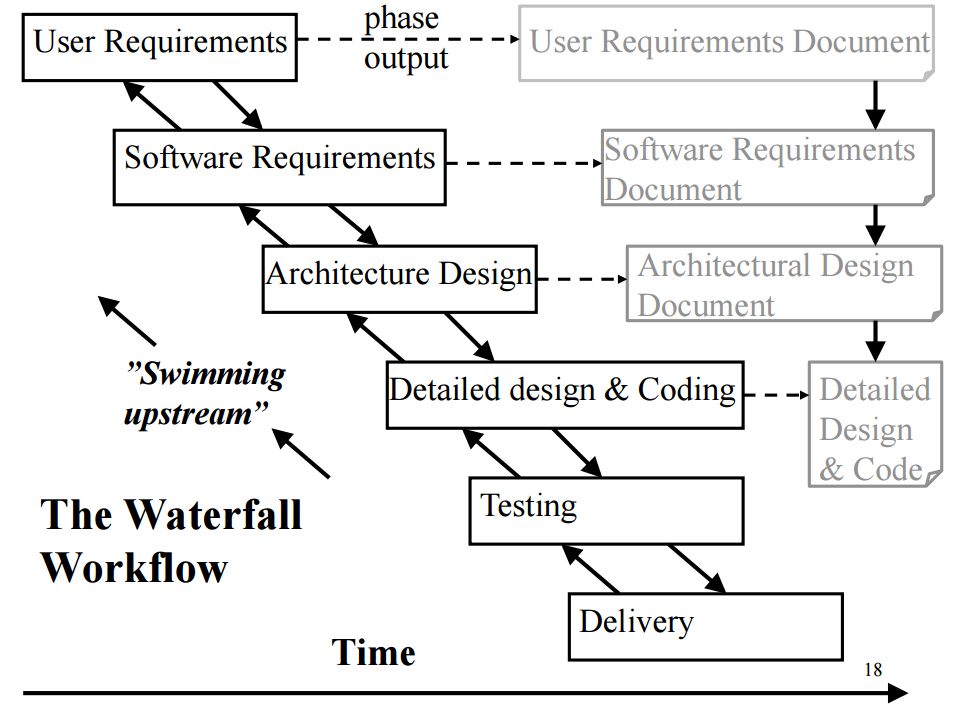
**Future Scope**

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.

PROJECT ORGANIZATION

**Software Process Models**

As the requirements were clear at first, and is not expected to change during the course of the project; and having a fixed outcome whose accuracy can be mathematical calculated, we decided to use the Waterfall Model.



**Roles and Responsibilities**

Algorithm Designer:

* Analyses various algorithms and selects the optimal ones
* Optimizes memory and run time complexity
* Decides on the mathematical models for computation

System Designer:

* Designs systems from a user perspective
* Designs usability of the application
* Designs application software components

Programmer:

* Prototypes, develops, and unit tests application software components or fragments

LITERATURE SURVEY

**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.

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 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.

**Main Body**

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 more algorithms have been developed for tackling the problem of multilabel classification. The most straightforward way is binary relevance (BR) learning; it constructs L binary classifiers, which are trained on the L labels independently. Thus, the prediction of the label set is composed of independent predictions for individual labels. However, labels often occur together, that is, the presence of a specific label may suppress or exhibit the likelihood of other labels. To address this limitation of BR, pairwise decomposition (PW) and label powerset (LP) approaches consider label dependencies during the transformation by either generating pairwise subproblems or the powerset of possible label combinations. Classifier chains are another popular approach that extend BR by including previous predictions into the predictions of subsequent labels. They present a large-margin classifier, RankSVM, that minimizes a ranking loss by penalizing incorrectly ordered pairs of labels. This setting can be used for multi-label classification by assuming that the ranking algorithm has to rank each relevant label before each irrelevant label. In order to make a prediction, the ranking has to be calibrated, i.e., a threshold has to be found that splits the ranking into relevant and irrelevant labels. Similarly, Zhang and Zhou introduced a framework that learns ranking errors in neural networks via backpropagation (BP-MLL).

**Conclusion of Survey**

The most prominent learning method for multi-label text classification is to use a Binary Relevance approach with strong binary classifiers such as SVMs despite its simplicity. It is well known that characteristics of high-dimensional and sparse data, such as text data, make decision problems linearly separable, and this characteristic suits the strengths of SVM classifiers well. Unlike benchmark datasets, real-world text collections consist of a large number of training examples represented in a high-dimensional space with a large amount of labels. To handle such datasets, researchers have derived efficient linear SVMs that can handle large-scale problems. The training time of these solvers scales linearly with the number of instances, so that they show good performance on standard benchmarks. However, their performance decreases as the number of labels grows and the label frequency distribution becomes skewed. In such cases, it is also intractable to employ methods that minimize ranking errors among labels or that learn joint probability distributions of labels.

Software Requirement Specification

**External Interface Requirements**

User Interfaces:

The user provides the input CSV file for the python based application. The result is then flushed locally to the disk as a text file. The user runs the python program from the terminal.

Hardware Interfaces:

* Minimum Requirements: 4GB RAM, 20GB Hard disk, I3 processor
* Recommended Requirements: 8GB RAM, 40GB Hard Disk, I7 processor

Software Interfaces:

The software is executed by running the python program on the terminal. Several python modules like Pandas, Scipy, numpy are needed on the client’s local machine.

Communication Interfaces:

* Input to program: CSV file containing details of the documents, terms and their corresponding categories.
* Output of program: The categories of the respective documents is flushed to the local disk in the form of a text file in the recommended format.

**Functional Requirements**

Functional Process Requirements:

It describes what the application must do. Process requirements relate to the entities and attributes from the data requirements to the users’ needs.

The application must be capable of performing the following tasks:

* Fetch the data from the input CSV file
* Parse data into the corresponding entities.
* Compute the TFIDF vectors for each document.
* Compute the centroid for each category.
* Given a new document and its term frequencies, the application must predict the category to which the document belongs by the nearest centroid approach.

**Software System Attributes**

Reliability:

The application’s reliability depends on the accuracy of its predictions. The predictions can be drastically improved by providing a larger training set in the CSV file.

Availability:

The program is available 24x7.

Security:

Since our program does not interact with any entity over the internet, it is highly secure. All the computations are performed locally on the machine.

Portability:

The software is fairly portable The client must install a few python dependencies on his/her local machine before running our application.

Maintainability:

The software has minimum maintenance requirements since it is prediction based.

Performance:

The performance directly depends on the RAM available on the local machine. This is because large computations are performed in memory in our software.

**Performance Requirements**

* The local machine must not be shut down/restarted while the computations are being performed by the application.
* The efficiency of the software depends directly on the RAM available on the local machine since all computations are performed in-memory.
* Missing data in the CSV file must be avoided. This would lead to incorrect predictions.

**Database Requirements**

The program does not make use of a conventional database to store the training data. The input to the program is in the form of a CSV file, and the output is written to a text file in the recommended format.

**Design Constraints**

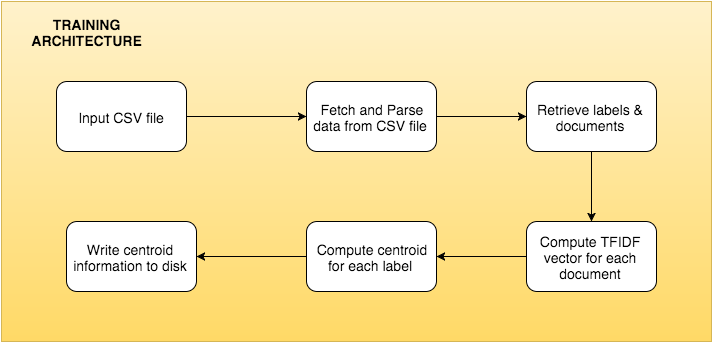
* Minimum RAM required: 4 GB
* Minimum hard disk required: 20 GB
* Minimum process required: i3

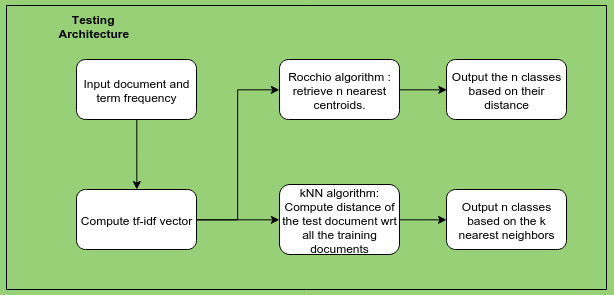
Design

**Introduction**

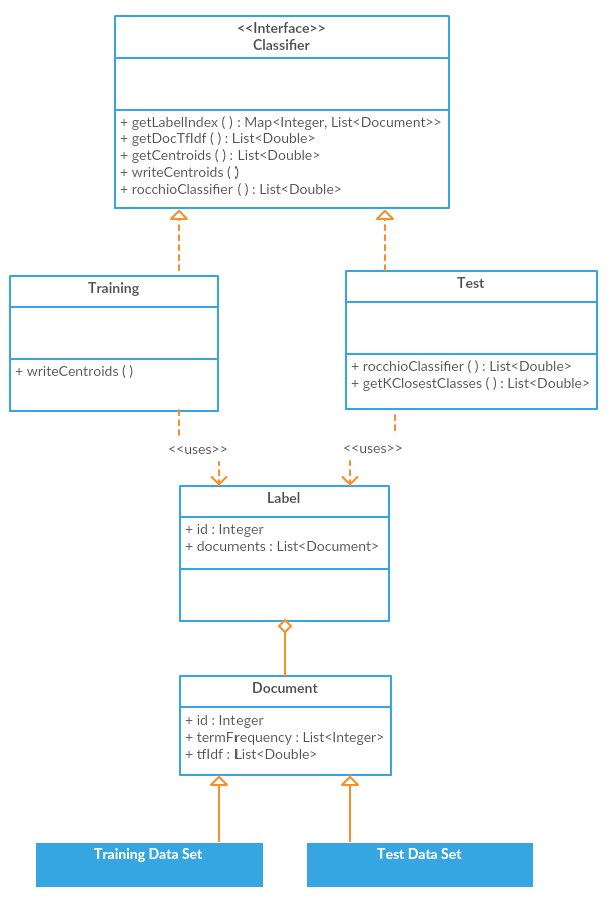
* Fetching of data
* Parsing of data
* Compute tf and idf
* Compute term frequency
* Compute inverse document frequency
* Compute tf-idf vector for each document
* Compute centroid for each category.
* Compute k nearest centroids for a given tf-idf vector
* Validate the predicted classification

**Architecture Design**

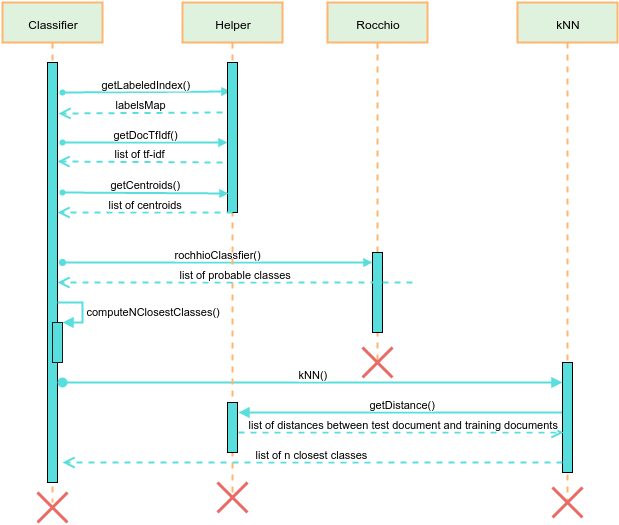
****

****

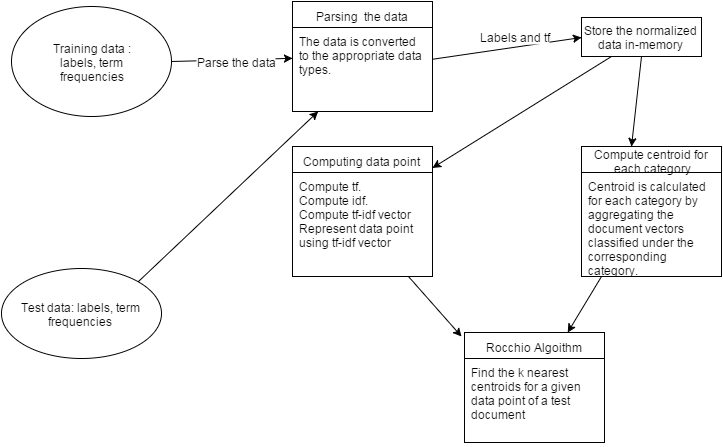
**Class Diagram**

****

**Sequence Diagram**

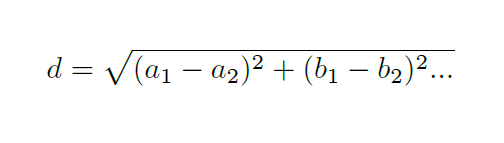
****

**Data flow diagram**

****

**Metric Calculation**

In this project, 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:



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.

IMPLEMENTATION

**Tools and Technology Used**

GPU:

A graphics processing unit (GPU), also occasionally called visual processing unit (VPU), is a specialized [electronic circuit](https://en.wikipedia.org/wiki/Electronic_circuit) designed to rapidly manipulate and alter memory to accelerate the creation of images in a [frame buffer](https://en.wikipedia.org/wiki/Frame_buffer) intended for output to a display. GPUs are used in [embedded systems](https://en.wikipedia.org/wiki/Embedded_system), [mobile phones](https://en.wikipedia.org/wiki/Mobile_phone), [personal computers](https://en.wikipedia.org/wiki/Personal_computer), [workstations](https://en.wikipedia.org/wiki/Workstation), and [game consoles](https://en.wikipedia.org/wiki/Game_console). Modern GPUs are very efficient at manipulating [computer graphics](https://en.wikipedia.org/wiki/Computer_graphics) and [image processing](https://en.wikipedia.org/wiki/Image_processing), and their highly parallel structure makes them more effective than general-purpose [CPUs](https://en.wikipedia.org/wiki/Central_processing_unit) for [algorithms](https://en.wikipedia.org/wiki/Algorithm) where the processing of large blocks of visual data is done in parallel. In a personal computer, a GPU can be present on a [video card](https://en.wikipedia.org/wiki/Video_card), or it can be embedded on the [motherboard](https://en.wikipedia.org/wiki/Motherboard) or—in certain CPUs—on the CPU [die](https://en.wikipedia.org/wiki/Die_(integrated_circuit)).

NVIDIA CUDA:

CUDA, which stands for Compute Unified Device Architecture, is a [parallel computing](https://en.wikipedia.org/wiki/Parallel_computing) platform and [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface) (API) model created by [NVIDIA](https://en.wikipedia.org/wiki/NVIDIA). It allows [software developers](https://en.wikipedia.org/wiki/Software_developer) to use a CUDA-enabled [graphics processing unit](https://en.wikipedia.org/wiki/Graphics_processing_unit) (GPU) for general purpose processing – an approach known as [GPGPU](https://en.wikipedia.org/wiki/GPGPU). The CUDA platform is a software layer that gives direct access to the GPU's virtual [instruction set](https://en.wikipedia.org/wiki/Instruction_set) and parallel computational elements.

The CUDA platform is designed to work with programming languages such as [C](https://en.wikipedia.org/wiki/C_(programming_language)), [C++](https://en.wikipedia.org/wiki/C%2B%2B) and [Fortran](https://en.wikipedia.org/wiki/Fortran). This accessibility makes it easier for specialists in parallel programming to utilize GPU resources, as opposed to previous API solutions like [Direct3D](https://en.wikipedia.org/wiki/Direct3D) and [OpenGL](https://en.wikipedia.org/wiki/OpenGL), which required advanced skills in graphics programming. Also, CUDA supports programming frameworks such as [OpenACC](https://en.wikipedia.org/wiki/OpenACC" \o "OpenACC) and [OpenCL](https://en.wikipedia.org/wiki/OpenCL" \o "OpenCL).

PyCUDA:

PyCUDA gives easy, Pythonic access to [Nvidia](http://nvidia.com/)‘s [CUDA](http://nvidia.com/cuda/) parallel computation API.

It’s advantages include -

* Object cleanup tied to lifetime of objects. This idiom, often called [RAII](http://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization) in C++, makes it much easier to write correct, leak- and crash-free code. PyCUDA knows about dependencies, too, so (for example) it won’t detach from a context before all memory allocated in it is also freed.
* Convenience. Abstractions like [pycuda.compiler.SourceModule](http://documen.tician.de/pycuda/driver.html#pycuda.compiler.SourceModule) and [pycuda.gpuarray.GPUArray](http://documen.tician.de/pycuda/array.html#pycuda.gpuarray.GPUArray) make CUDA programming even more convenient than with Nvidia’s C-based runtime.
* Completeness. PyCUDA puts the full power of CUDA’s driver API at your disposal, if you wish.
* Automatic Error Checking. All CUDA errors are automatically translated into Python exceptions.
* Speed. PyCUDA’s base layer is written in C++, so all the niceties above are virtually free.

**Data Set Description**

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:

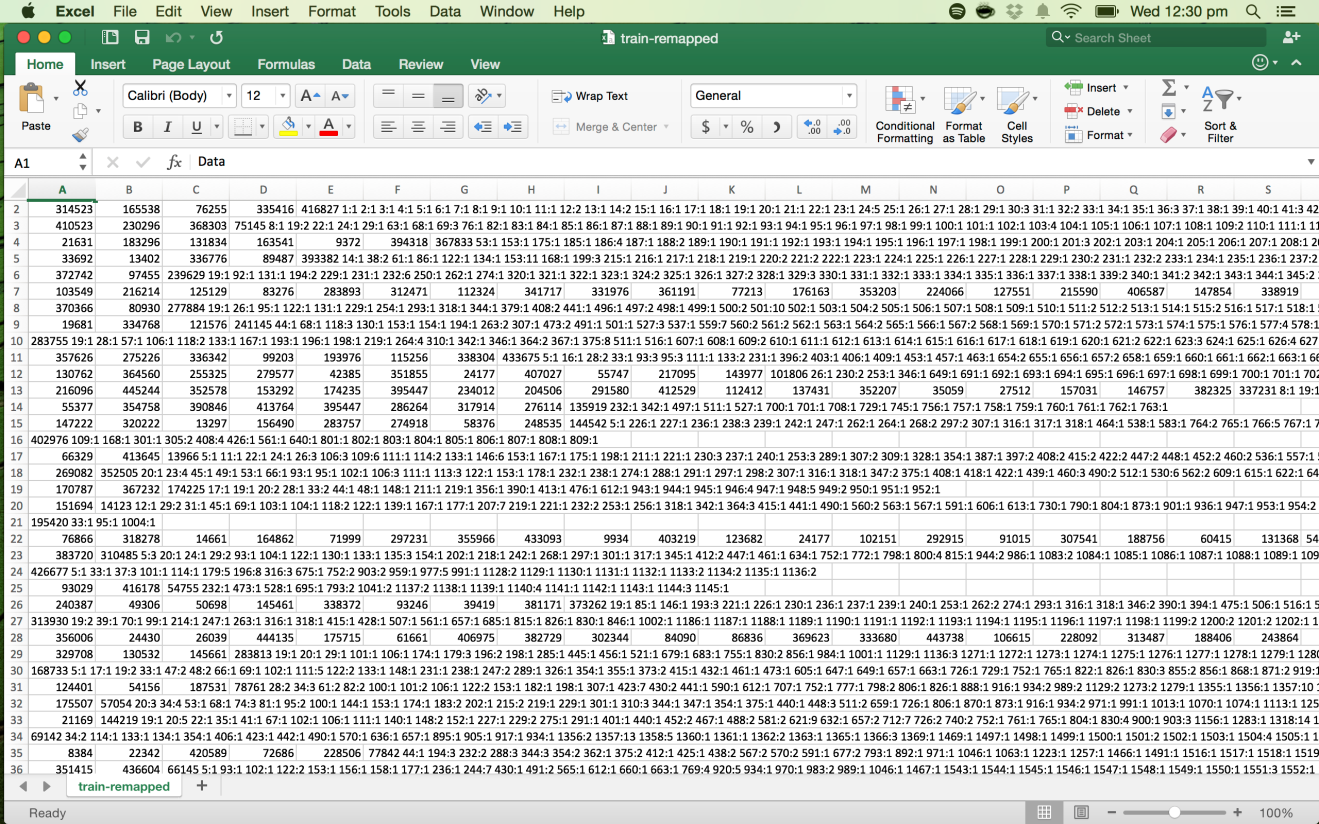
347 73

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.



**Overall Project View**

* Fetching of data
* Parsing of data
* Compute tf and idf
* Compute term frequency
* Compute inverse document frequency
* Compute tf-idf vector for each document
* Compute centroid for each category.
* Compute k nearest centroids for a given tf-idf vector
* Validate the predicted classification

**Algorithms Used**

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.

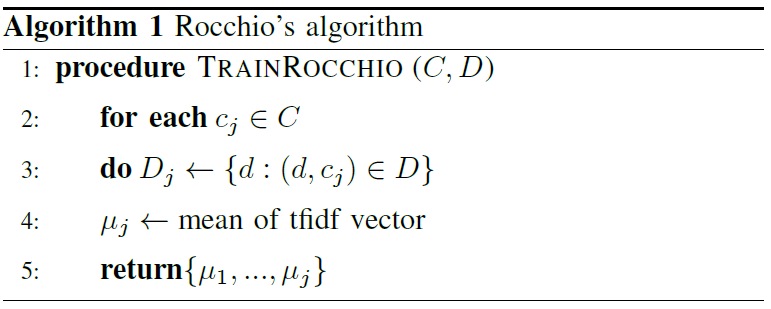
When used in text categorization, the nearest centroid classifier is also known as the Rocchio classifier because of how similar it is to the Rocchio algorithm for relevance feedback. An elongated version of the nearest centroid classifier has found many uses in the medical domain, specifically classification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

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.

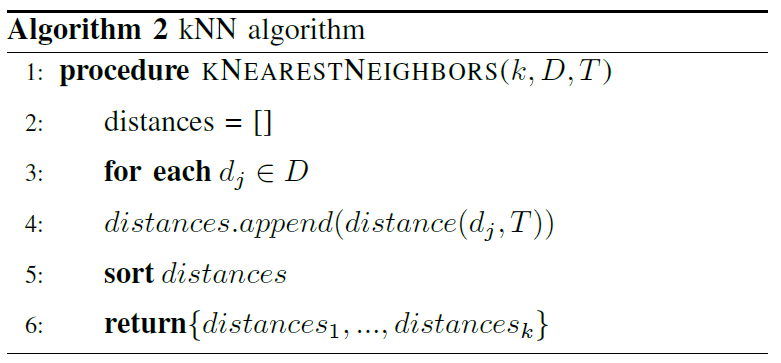
For the classification of a test document, we use parallel programming. We run the prediction part of the algorithm on a GPU, using the CUDA platform. We compute the distances of each of the centroid with respect to the test vector parallely. Since the computation of the distance vectors are independent 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.

Rocchio Algorithm:



kNN Algorithm:



**Modules Description**

**Fetching of data**

The training data is in the libSVM format. Each line corresponds to a sparse document vector abs has the following format:

label, label … feat:value … feat:value

labelis an integer corresponding to a category. Feat is an integer representing a term and value is a double representing the term frequency.

**Parsing of data**

The labels, terms and corresponding term frequency will be parsed into appropriate data types required for the future computation of nearest centroid for a given test document vector.

**Compute tf and idf**

Compute the inverse term frequency (tf) for each term in the entire vocabulary of the training set. Compute the term frequency (idf) for each term in a document. Using these two values, compute the tf-idf value for each term in the document.

**Compute tf-idf vector for each document**

Represent the document as a vector using the tf and idf values computed from the previous value.

**Compute centroid for each category**

For each class, aggregate the term frequencies of all terms existing in the documents classified under that class. Calculate the idf for each of the terms existing in the documents classified under that class. Calculate the centroid for the class from these values.

**Compute k nearest centroid for a given tf-idf vector**

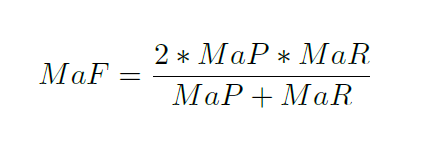
We use the nearest centroid classifier to classify a test document to k classes, such that the data point representing the document is closest to the k centroids chosen.

**Validate the predicted classification**

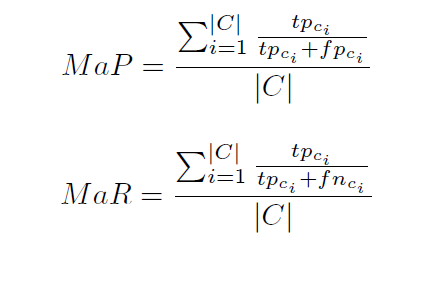
We submit our classified data to the kaggle platform, which would run our data over the correct data and give us the accuracy of our classification.

RESULTS

The evaluation metric for the Kaggle Competition is as follows:

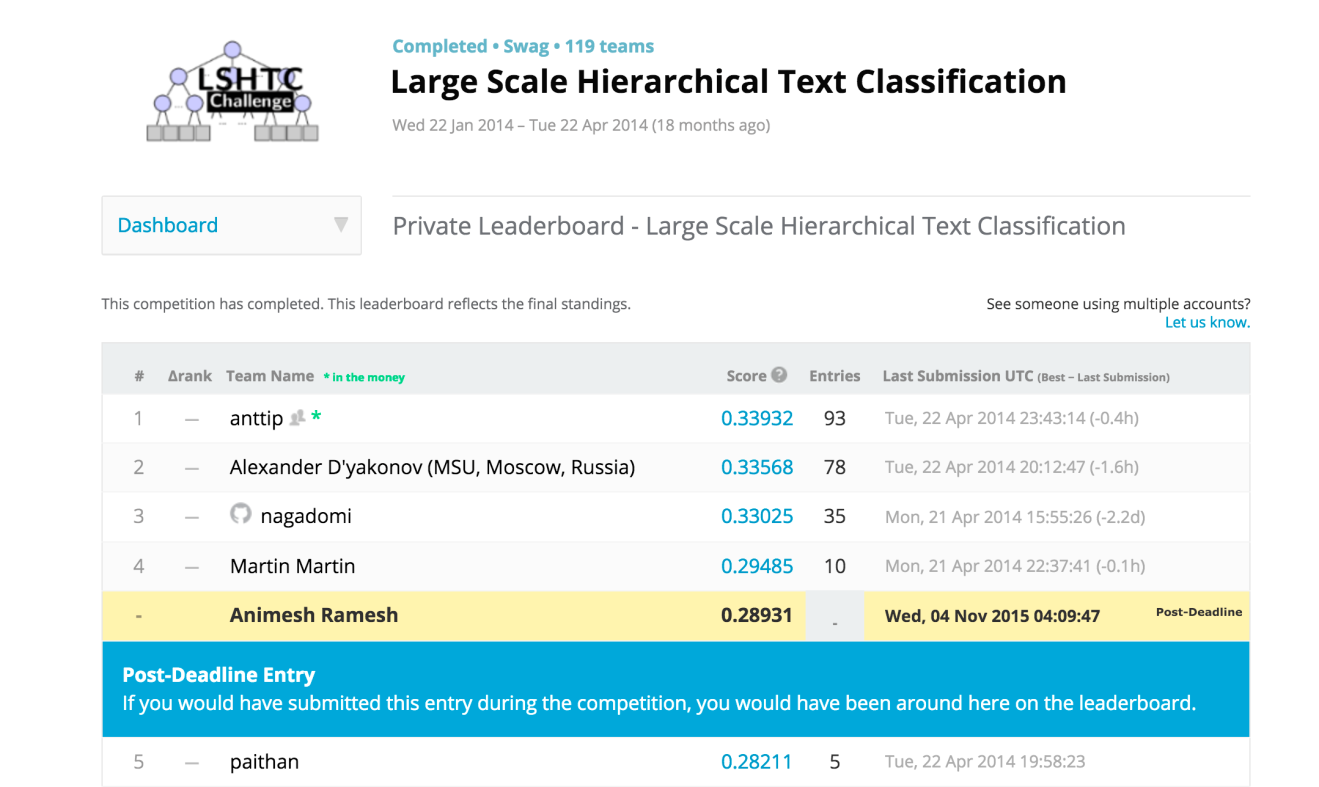


For a set of classes C{c1,…,ck} macro precision and macro recall are calculated as follows:



where tp{ci}, fp{ci}, fn{ci} are the true positives, false positives and false negatives respectively for class {ci}.

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.



CONCLUSION AND FUTURE SCOPE

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.

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.

REFERENCES

* Kaggle LSHTC Challenge, http://kaggle.com/c/lshtc
* Krzysztof Sopyła, Paweł Drozda, Przemysław Górecki. SVM with CUDA Accelerated Kernels for Big Sparse Problems. 2012
* V. Vaitheeshwaran, Kapil Kumar Nagwanshi, T. V. Rao. Multicore Processing for Classification and Clustering Algorithms, 2012.
* Basu, T. Effective Text Classification by a Supervised Feature Selection Approach 2012
* A Sun et. al. Short text classification using very few words. 2012
* 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
* YS Lin, JY Jiang, SJ Lee - Knowledge and Data Engineering. A similarity measure for text classification and clustering. 2014
* AK Uysal, S Gunal - Knowledge-Based Systems. A novel probabilistic feature selection method for text classification. 2012
* JY Yoo, D Yang. Classification Scheme of Unstructured Text Document using TF-IDF and Naive Bayes Classifier. 2015
* M Ghiassi, M Olschimke, B Moon, P Arnaudo. Automated text classification using a dynamic artificial neural network model. 2012
* P. McCullagh, J. Yang, How many clusters? Bayesian Analysis 3 (2008)
* J Read, B Pfahringer, G Holmes, E Frank. Classifier chains for multi-label classification. 2011
* C Bielza, G Li, P Larranaga. Multi-dimensional classification with Bayesian networks. 2011
* J Read, A Bifet, G Holmes, B Pfahringer. Scalable and efficient multi-label classification for evolving data streams. 2012
* L Zhou, Z Yu, J Lin, S Zhu. Acceleration of Naive-Bayes algorithm on multicore processor for massive text classification. 2014
* S Canuto, T Salles, MA Gonçalves. On Efficient Meta-Level Features for Effective Text Classification. 2014