Induction From Multi-label Examples

Hind Hazza Alsharif[1], Wadee Saleh Alhalabi[2], Miroslav Kubat[3]

[1, 2] Computer Science Department
Faculty of Computing and Information Technology
King Abdul Aziz University
[3] Electrical and Computer Engineering Department
University of Miami

Abstract: The task of text categorization is to assign one or more classes to a document. The simplest machine learning approach to such domains, simply induces a binary classifier separately for each class, and then uses these classifiers in parallel. An example of motivating application is a digital library collection that used to be classified into classes and sub-classes in a hierarchical order. Another important issue that we are considering is the document might belong to more than one class, in this case we will be working on a high performance multi-class label classifier. The study we are intending to do herein is going to show how much we can gain from machine learning. This mean, if we need something like 10 to 15% of the data for training, and testing or do we need > 50% of the data set for training and testing. In the latter case, the machine learning may don't contribute that much. However, if 10 to 15% of the data set is needed, then, machine learning has a great contribution. The last issue we are working on in this research is the inter-class relation. Which means, if the example is classified to belong to a class C, does this mean, the example belong to parents and grandparents classes of the class C, and on the opposite way too? We will use a framework to classify documents automatically and this can indeed answer these questions.

[Hind Hazza Alsharif, Wadee Saleh Alhalabi, Miroslav Kubat. **Induction From Multi-label Examples**. *Life Sci J* 2014;11(10):495-511] (ISSN:1097-8135). http://www.lifesciencesite.com. 67

Keywords: text categorization, induction process, multi-label classifiers, inter-class relation, KNN algorithms, Naïve Bays algorithms

1. Introduction

The main goal of a classification induction process is to find the mechanism (rules) able to place an example or a stream of examples into sets of categories called classes. In the case of multi-label classification induction, an example is allowed to belong to more than one class at a time, and the classes are hierarchically ordered. This is referred to as Hierarchical Multi label Classification (HMC). The classification of a library collection (where book titles represent the examples and each of the scientific field represents a class) is an example of an HMC problem. The class-to-class relations are defined by a Directed Acyclic Graph (DAG) (Figure1.1) which indicates that there are no cycles. The nodes and the edges define the structure of the network, and the conditional probabilities are the parameters to give the structure to the graph [1].

The focus of this research is on the HMC problem, with emphasis on several case studies used for drawing observations and reaching general conclusions. Aiming to build a proper induction system for these problems, the top down approach was preferred. It started by inducing a classifier for each class of the highest level of the DAG and continued downward by employing the higher-level classifiers when creating the training sets for lower-level classifiers.

$$\begin{tikzcd}
 & & 0 \arrow[dl] \arrow[dr] & & \\
 & 1 \arrow[dl] \arrow[d] & & 2 \arrow[dl, dashed] \arrow[dr] & \\
1.1 & 1.2 \arrow[r, dashed] & 2.1 & & 2.2 \arrow[dl] \arrow[d] \\
 & & & 2.2.1 & 2.2.2
\end{tikzcd}$$

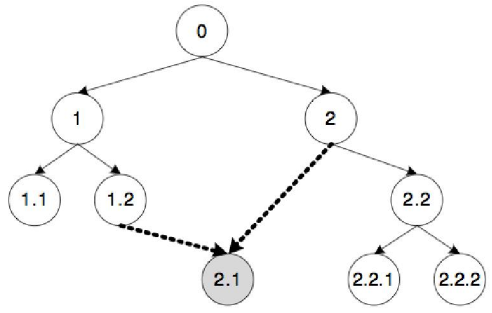

Figure 1.1: An example of a DAG-structured class hierarchy as presented in [2]

The scope was to develop a proprietary methodology and algorithm and compare it with a couple of many popular algorithms including Support Vector Machines (SVM) [3-5], K-Nearest Neighbor (KNN) [6] and Naïve Bays [7]. The comparative study consisted in: i) classifying examples into hierarchically ordered classes and ii) finding their inter-class relation.

As the work in this research progressed and as recommended by [2], we realized that HMC's performance has to be evaluated along somewhat different criteria than those used in classical machine learning. For example, let C be a set of classes to which an example X belongs to. A perfect classifier will label X with all classes from C, never suggesting any class from outside C; moreover, an HMC usually requires that any X that has been labeled with Ci should also be labeled with all ancestors of Ci in the class hierarchy. To be able to reflect these requirements in performance evaluation, we used an adequate extensions of precision and recall introduced by Clare et al. [8].

1.1. Problem Statement

A graph mainly consists of a set of nodes, N, and a set of edges, E, where an edge is an ordered pair of nodes, (N_p, N_c) $\in$ E $\subseteq$ {N × N }. $N_p$in this pair, is known as a parent, and Nc as a child. A path (Na → Nc) from an ancestor (Na) to a child (Nc) is referred to be a series of edges, {(N_1, N_2), (N_2, N_3), . . . , (N_{n-1}, N_n)} in a way that N_1 = Na and Nn = Nc. The existence of a path in a DAG, is Na → Nc, guarantees the non-existence of the opposite-direction path, Nc → Na. A leaf node is known to be a node without any child, and a root node is known to be a node without any parent.

In this research, this problem is addressed by considering a set of class labels, C, whose mutual relations are specified by a class hierarchy, H, which has the form of a DAG in which each node represents only one class.

X $\subseteq$ R^p is a finite set of examples, each described by a set of p numeric attributes. We assume that each $x_i \in$ X is assigned a set of class labels, L = {C_1 , ..., C_i } $\subseteq$ H (all classes belong to the given class hierarchy). An example belonging to class C_c is assumed to also belong to all C_c's ancestor classes, C_a. This property is called "hierarchical constraint".

There are two versions of the hierarchical classification task: i) the Mandatory Leaf-Node Problem (MLNP), where only the leaf-node classes are used and; ii) the Non-Mandatory Leaf Node Problem (NMLNP), where an example can be labeled with any class from the given class hierarchy. Considering the class hierarchy from Figure 1.1, MLNP permits an example to be labeled only with a subset of {C1.1, C2.1, C2.2.1, C2.2.2}, but NMLNP allows also the other class labels (e.g., C1 or C2.2). This research focuses on the general NMLNP, because we assign examples to any node in the hierarchy.

1.2. Two main questions were addressed in this research:

Suppose that a machine learning algorithm has already induced classifiers for some highest-level classes. Does this facilitate any future attempts at the induction of lower-level classes? For instance, if an example was classified to a lower level class, can this example belong to the parent and grandparent classes?

Turning this upside down, suppose we know the lowest-level classes. Can this be exploited in the induction of the parents of these classes? For instance, if an example was classified in the upper level class, can this example be a parent of the lower level classes?

Aiming to answer these questions, a proprietary algorithm will be built. It will test whether an example is classified into its corresponding child and grandchild, as well as if the grandchild is belonged to its accurate parent and grandparent. The focus is on the inter-class relations and we want to look at the parent-child and child-parent relations, this aspect representing the main contribution of this study.

The significance of the research is the motivation for the use of machine learning in digital libraries which can be defined as follow: The digital library needs to be able to identify all documents relevant to a user's query. This function is sometimes supported by an indexing system in which each document is tagged with the labels of all the topics it represents. The indexing system is relatively easy to create in a small collection: an expert reads each single document, and then decides which topics it represents. In large collections, this might be expensive and clearly impossible if hundreds of thousands of documents are added to the library every week, or even on a daily basis. In this latter case, one solution is to classify manually only a subset of the documents, and then employ the training set, to obtain the induction of a classifier. The induced classifier then labels those documents that have not been classified manually. The principle can be applied to other domains, not just digital libraries.

The main research question is the following: how many documents should we classify manually if we want to induce a high-performance classifier? To put the question in another format: How much can be gained from the use of machine learning? For example, suppose we have 106 documents. If we manually classify only a few, the induced classifier will over fit the training examples, and thus perform poorly on the remaining documents. The situation will improve if the training set consists of about 10% of the collection or more; but then, the price of manual classification will become prohibitive. This motivates an experimental study whose goal is to identify the right size of the training set, and this is what we want to do in this research.

1.3. Possible conclusions:

It may turn out that only a small percentage of all examples are enough for the induction of a relatively high-performance classifier. In this case, the use of machine learning is justified. Conversely, it may turn out that even using 50% of the examples for training is not enough. In this case, machine learning does not seem to help. Most likely, the observed result will be somewhere between these two extremes. We might want to verify if the observation is the same in each of the studied experimental domains. This means, we want to repeat this experiment for several different domains.

2. Background

Over the past few years, studies of induction from multi-label examples have targeted two specific strategies: induction of sets of binary classifiers, and induction of one large multi-label classifier. For the induction of sets of binary classifiers, mechanisms based on Bayesian theory were studied by Friedman et al. [7], and McCallum and Nigam [9]. The latter was investigated by Baoli et al. [6], and the currently popular SVM were discussed by Joachimis [4] and Kwok [10]. Unfortunately, binary classifiers ignore inter-class relations, which sometimes lead to performance degradation. In this study, the focus is on these inter-class relations.

2.1. Bayesian Networks:

The Naïve Bays, classifier learns from training data the conditional probability of each attribute A_i given the class label C. Classification is then done by applying Bayes rule to compute the probability of C given the particular instance of A_1, . . . , A_n, and then predicting the class with the highest posterior probability. This computation is rendered feasible by making a strong independence assumption: all the attributes A_i are conditionally independent given the value of the class C. By independence means the probabilistic independence, that is, A is independent of B.

A naive Bayesian classifier has the simple structure shown in Figure 2.1. This network captures the main assumption behind the naive Bayesian classifier, namely, that every attribute (every leaf in the network) is independent from the rest of the attributes, given the state of the class variable (the root in the network). Thus, it is said that the performance of naive Bayes is somewhat surprising due its dependency [9].

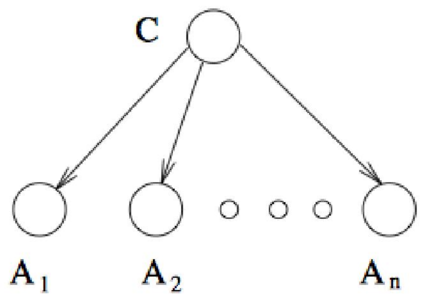

Figure 2.1: The structure of the naive Bayes network.

2.2. K-Nearest Neighbor:

In a text categorization system based on the K-Nearest Neighbor algorithm (KNN), k is the most important parameter. To classify a new document, the k-nearest documents in the training set are determined first. The prediction of categories for this document can then be made according to the category distribution among the k nearest neighbors. Generally speaking, the class distribution in a training set is not even; some classes may have more samples than others. The system's performance is very sensitive to the choice of the parameter k. And it is very likely that a fixed k value will result in a bias for large categories, and will not make full use of the information in the training set [6]. Baoli et al. [6] studied a text categorization system based on the KNN, and an improved KNN strategy (in which different numbers of nearest neighbors for different categories are used instead of a fixed number across all categories) was proposed.

2.3. Support Vector Machine:

SVM aims to fit an Optimal Separating Hyperplane (OSH) between classes by focusing on the training samples that lie at the edge of the class distributions, the support vectors. The OSH is oriented such that it is placed at the maximum distance between the sets of support vectors, which leads to generalize more accurately and aims to minimize the training error such as neural networks [26]. Joachims [4] introduced the Support Vector Machine (SVM) for text categorization from examples by analyzing particular properties of learning with text data. Practical results showed that SVM's achieved good performance on text categorization tasks, substantial improvements over the currently best performing methods being observed. In addition, SVMs are fully automatic, and they eliminate the need for manual parameter tuning making the text categorization process much easier.

2.4. Hierarchical Multi-label Classification Strategies

Silla and Freitas [29] explored the solutions to the HMC problems and presented three fundamental strategies: 1) flat classification, 2) top-down approach "local classification", 3) the "big-bang" approach or global classification.

2.5.1 Flat classification

The advantage of this strategy is that it enables the use of traditional machine-learning techniques such as neural networks, decision trees, or SVM to be implemented in the HMC as reported by [30-32]. Basically it ignores the class hierarchy and deals only with the leaf-node classes (as if the problem were MLNP), whether by a single multi-label classifier or by a set of binary classifiers (a separate one for each leaf node). If the leaf-node class label is known for each example, this strategy is possible. Besides, if the nature of the application seems to allow the user to afford the inability to identify non-terminal classes.

2.5.2 Top-down approach (local classifier):

The most common approach in HMC induction is the local classifier. In the simplest scenario, for each node in the DAG-specified class hierarchy, a separate (local) classifier is induced, and the processing is started by creating a whole hierarchy of classifiers, from top levels going downwards.

The main advantage of this method is simplicity. On the other hand, the approach tends to suffer from "error propagation", which means that misclassifications of the higher- level classes are propagated to the lower levels.

The first experiments with this approach were provided by Koller and Sahami [14] by choosing Naive Bayes to induce each individual class. The authors experimented with tree-structured class hierarchies with no more than one parent for any node and limited to just two levels.

Fagni and Sebastiani [78, 79] compared four different policies (Sibling, ALL, BestGlobal, and BestLocal) to generate a set of binary training data. Tree-structured hierarchical versions of boosting and SVM called TreeBoost and TreeSVM were used. The best results were obtained with the Sibling policy in which the negative training examples of the ith node are all positive examples of its Sibling nodes in the hierarchy.

This strategy was applied to text classification by Sun and Lim [33], where the class hierarchy was a plain tree structure. They induced two SVMs for each class: a local classifier and a sub-tree classifier. An example is labeled as c_i by the local classifier, while the sub-tree classifier decides whether or not this example should be passed to ci's sub-classifiers. This approach was extended to domains with DAG-structured class hierarchies, by Nguyen et al. [34], the DAG hierarchy being transformed into a set of tree hierarchies. Experimental results indicated high classification performance as well as high computational costs.

Looking to further improve the performance, Secker et al. [35] used several induction algorithms for each node of the hierarchy: Naive Bayes, SMO, 3-NN, etc. Ten classifiers were trained for each node, and the one with the best classification results was selected. This improved classification accuracy, but the computational costs were even higher than in the previous attempt.

Bi and Kwok [36] applied the Kernel Dependency Estimation (KDE) to reduce the number of classes in the hierarchy during the training process. This is because the number of classes in the hierarchy is usually unmanageable. The authors proposed an algorithm called "Condensing Sort and Selection Algorithm (CSSA)" for the tree structured hierarchies and, then, extended it to the CSSAG algorithm for the DAG-structured hierarchies. However, they did not report experimental results regarding induction time and the number of reduced classes.

Alaydie et al. [37] proposed a framework called "HiBLADE (Hierarchical multi-label Boosting with Label Dependency)," applied to tree-structured hierarchies. The classifier for each class is a boosting-type algorithm, such as ADABOOST, where the new model for each boosting iteration is updated by utilizing the proposed Baysian correlation.

2.5.3. The "big-bang" approach (global classifier):

Some authors preferred to induce one big (global) classification model to cover the entire class hierarchy, instead of inducing a separate binary classifier for each node. In this manner, mutual interdependencies of the classes can be easily taken into account, and the global classifier is often smaller than the total of the local classifiers.

Clare and King [8] developed a hierarchical extension to the decision-tree generator C4.5 [38] and applied it to functional-genomics data. Their system is known as HC4.5, a mechanism for weighing the entropy formula (in order to give higher priority to more specific classes) being induced.

Seeking to make the decision-tree paradigm applicable to hierarchical domains, an attempt was reported by Blockeel et al. [39] whose Clus-HMC is a hierarchical version of the earlier "predictive clustering tree" (PCT) [40]. Ven et al. [3] improved Clus-HMC so it could be used in DAG-specified class hierarchies. Schietgat et al. [12] proposed an ensemble version of the algorithm Clus-HMC-ENS.

Although the ensemble concept can improve classification accuracy, its computational costs are much higher than those of the original Clus-HMC.

A global-approach hierarchical framework based on the K-Nearest Neighbor classifier (k-NN), was proposed by Pandey et al. [41]. There system's improvements include: i) a Lin's semantic similarity measure used as a distance measure; ii) the prediction function of the i-th class incorporates the inter-relationship score of the i-th class to other classes in the hierarchy; and iii) the mechanism to filter insignificant class inter-relationships was suggested.

Lo et al. [42] proposed a basis expansion model for multi-label classification, where a basis function is a Label Power set (LP) classifier trained on a random k-label set. LP [43] method is a multi-label learning algorithm which basically reduces the multi-label classification problem to a single-label multi-class classification problem by dealing with each distinct combination of labels in the training set as a different class. Random k-Label sets (RAKEL) [44] has introduced to overcome the drawback of the LP method. It randomly selects a number of label subsets from the original set of labels and then uses LP for training the corresponding multi-label classifiers. Experiments were conducted on ten benchmark datasets belonging to different domains, including: scene, enron, cal500, major miner, medical bibtex, and four versions of delicious (from dlc1 to dlc4). More details on these data sets are available at the MULAN library website [45].

Qu et al. [46] proposed a Multi-Label classification algorithm based on label-Specific Features (MLSF). The feature density on the positive and negative instances set of each class was first computed and after that, the features of high density from the positive and negative instances set of each class were selected. The intersection was taken as the label-specific features of the corresponding class. Finally, the multi-label data was classified on the basis of label-specific features. The classifiers induction process of MLSF is similar to the original binary classifiers. Given an unlabeled instance xu ∈U , the feature sets for each class label are first rebuild based on the label-specific features, and then the corresponding classifier is used to predict whether it has the label or not. The proposed MLSF is compared with three multi-label learning algorithms, including ML-KNN, LIFT, and Rank-SVM. The experiments were employed on both regular-scale and large-scale. For the results, common evaluation criteria for multi-label classification were used (hamming loss, one-error, coverage, and average precision). It is observed, that the performance of MLSF is comparable to that of LIFT on the regular-scale data sets and large-scale data sets and that MLSF and LIFT algorithms perform significantly better than ML-KNN and Rank-SVM.

Kong et al. [47], used the heterogeneous information networks to simplify the multi-label classification process. They focused on extracting the relationships among different class labels and data samples by mining the linkage structure of heterogeneous information networks. These relationships can be then used to effectively infer the correlations among different class labels in general, as well as the dependencies among the label sets of data examples that are inter-connected in the network. The proposed multi-label collective classification algorithm (called PIPL) was tested on a bio-informatic dataset SLAP [48], which is a heterogeneous network contains integrated data related to chemical compounds, genes, diseases, side effects, pathways etc.

2.6. Other Classification Methods

Other existing multi-label classification methods may include: BSVM (binary SVM), ECC (multi-label classification + ensemble); PISl (binary decomposition + meta-path based instance correlation):a collective classification approach [49], where instance correlations are from heterogeneous network; Icml (simple label correlation + instance correlation in homogeneous network): this method was proposed by Kong et al. [50, 50] which exploit relational features for inter-instance dependencies based on homogeneous network for multi-label collective classification; PIml (simple label correlation + meta-path based in- stance correlation): a multi-label collective classification approach extended from PIsl [49] by adding relational features according to inter- instance-cross-label dependencies for multi-label collective classification [50]; PIPL (meta-path based instance and label correlation): a method for multi-label collective classification in heterogeneous information networks. The only difference between PIPL and PIml is that PIml does not consider the meta-path based label correlation.

2.7. Performance Evaluation

In order to evaluate the multi-label classifiers, different methods than the ones used in the case of single-label problems are used because an example can be partially correct or incorrect [57]. According to [43], the measures used for evaluation of multi-label classification can be organized into two classes: i) bipartition based (includes example based measures and label based measures) and ii) ranking based (evaluates measures based on the ground truth of multi-label dataset). The example based measures evaluate the bipartitions over all examples of the evaluation dataset, while the label based measures divide the evaluation process into evaluations of each label [57].

In classical machine learning, the classifiers are usually evaluated by error-rate estimates. This error is obtained by the comparison between testing examples' with a pre-determined class labels with those class labels recommended by the classifier. This, however, is not quite enough when dealing with domains where one class significantly outnumbers the other [58]. For instance, if only 1% of the examples are positive, then a classifier that labels all examples as negative will achieve 99% accuracy.

For this latter case, other criteria are used, the most popular among them being precision and recall. Let us denote by TP the number of true positives, by FN the number of false negatives, by FP the number of false positives, and by TN the number of true negatives. Precision and recall (which are example based measures) are defined as follows:

$$Pr = \frac{TP}{TP+FP} \quad (1)$$

$$Re = \frac{TP}{TP+FN} \quad (2)$$

Precision is the percentage of truly positive examples among those labeled as such by the classifier; recall is the percentage of positive examples that have been recognized as such ("recalled") by the classifier. Which of the two is more important depends on the given domain. In order to combine them in a single formula, [59] proposed Fβ, where the user-specified parameter, $\beta \epsilon [0,\infty)$, quantifies each component's relative importance:

$$F\beta = \frac{(\beta^2+1)\times Pr \times Re}{\beta^2 \times Pr+Re} \quad (3)$$

It would be easy to show that β > 1 apportions more weight to recall while β < 1 emphasizes precision. Moreover, F β converges to recall if $\beta \rightarrow \infty$, and to precision if β = 0. If we do not want to give more weight to either of them, we use the neutral β = 1:

$$F_1 = \frac{2 \times Pr \times Re}{Pr+Re} \quad (4)$$

All this, however, applies only to domains where each example is labeled with one and only one class.

F-measure is the harmonic mean between precision and recall [52]:

$$F-measure = \frac{1}{d}\sum_{j=1}^{d} \frac{2p_j r_j}{(p_j+r_j)} \quad (5)$$

where p_j and r_j are the precision and recall for C_j. Here, the F-measure is calculated per label and then averaged.

Yang [60] proposed two methods to average the above metrics over multiple classes: (1) macro-averaging, where precision and recall are first computed separately for each class and then averaged; and (2) micro-averaging, where precision and recall are obtained by summing over all individual decisions. Which of the two approaches is better depends on the concrete application. Generally speaking, micro-F_1 weighs the classes by their relative frequency, whereas macro-F_1 gives equal weight to each class. The formulas are summarized in Table 2.1, where Pr_j, Re_j, and F1.j , stand for precision, recall, and F1 for the jth class (from l classes).

Table.1 Macro-averaging and micro-averaging of the performance criteria on the data set with l classes [60].

Average	Criteria	Equation
Macro	Precision	$Pr^M = \frac{\sum_{j=1}^{l} Pr_j}{l}$
	Recall	$Re^M = \frac{\sum_{j=1}^{l} Re_j}{l}$
	F_1	$F_1^M = \frac{\sum_{j=1}^{l} F_{1,j}}{l}$
Micro	Precision	$Pr^\mu = \frac{\sum_{j=1}^{l} TP_j}{\sum_{j=1}^{l}(TP_j+FP_j)}$
	Recall	$Re^\mu = \frac{\sum_{j=1}^{l} TP_i}{\sum_{j=1}^{l}(TP_j+FN_j)}$
	F_1	$F_1^\mu = \frac{2\times Pr^\mu \times Re^\mu}{Pr^\mu+Re^\mu}$

Hamming loss (an example based measure) [61] evaluates how many times an example-label pair is misclassified, i.e., label not belonging to the example is predicted or a label belonging to the example is not predicted. The smaller the value of hamming_loss(h), the better the performance. The performance is perfect when hamming_loss(h) = 0. This metric is defined as:

$$hamming_{loss\,(h)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{Q}\left|h(x_i)\Delta y_i\right| \quad (6)$$

where Δ stands for the symmetric difference between the two sets, N is the number of examples and Q is the total number of possible class labels. Yi denotes the set of true labels of examples xi and h(xi) denotes the set of predicted labels for the same examples.

3. Methodology

Digital libraries provide a huge range of information including text, movies, speeches, images, photos, books and others. This digital data provides large collections of content which naturally leads to the need of powerful tools that efficiently process, analyze, navigate, and browse the digital data [65]. Therefore, in this work, different data sets from books digital libraries and other contents were used. There are many digital libraries available online such as, Internet archive [66], Google books [67], Open library [68], The New York public library [69], and Wiley online library [70]. Library collections such as Wiley Online Library [70], which hosts the world's broadest and deepest multidisciplinary collection of online resources covering life, health and physical sciences, social science, and the humanities, Routers-21578 [71], which a collection appeared on the Reuters newswire in 1987. From the available sources Wiley Online Library [70], Routers-21578 text categorization collection data set [71], and the 20 Newsgroups data set [17] were chosen for the conducted experiments. Wiley Online Library hosts the world's broadest and deepest multidisciplinary collection of online resources covering life, health and physical sciences, social sciences, and humanities. It delivers seamless integrated access to over 4 million articles in 1500 journals, over 14,000 online books, and hundreds of reference works, laboratory protocols and databases. The documents in Routers-21578 [71] are organized and indexed with categories by personnel from Reuters Ltd. In 1990, Reuters and CGI made the documents available for research purposes to the Information Retrieval Laboratory of the Computer and Information Science Department at the University of Massachusetts at Amherst. There are multiple categories, and there are relationships among the categories, therefore are many possible feature sets can be extracted from the text.

The 20 Newsgroups data set [17] is a collection of approximately 20,000 newsgroup documents, partitioned across 20 different newsgroups, each corresponding to a different topic. It has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.

Since the data set that is provided by the digital library is considered as raw data, it may contain nominal attributes (un-necessary). Nominal attributes are defined by providing a <nominal-specification > listing the possible values: {the, for, in, on, edition, processes, systems...}. Also, a raw data set may contain many values that may be missing, so it is necessary to do some pre-processing. Once pre-processing was finished, a proprietary algorithm for multi-label class was implemented and compared with some existing algorithms.

3.1. Data pre-processing:

Since the data set is a raw data, it may contain many values that might be missing, so it is necessary to do some pre-processing. This phase consists of the following: i) data cleaning; ii) feature extraction; and iii) nominal to numerical conversion.

3.1.1. Data cleaning:

Removing un-necessary and meaningless words such as “introduction”, “handbook”, “edition” etc., is done in this stage. Its role is to reduce the dimensions of the dataset and to eliminate the elements that can create errors in the classification algorithm.

Meaningless words with very high frequency are considered as stop words [16], and these words are added to the Stop Word list. Removing such words will result in better results and it will not affect the classification efficiency at the same time.

In data cleaning, mainly the input file is parsed line by line and each line is being split into words by space character as a delimiter. Then each is getting its stem using the Porter stemming algorithm [15].

The Porter stemming algorithm is a process for removing the commoner morphological and inflexional endings from words in English. It is mainly used is as part of the normalization process that is usually done during processing information retrieval systems. After the stemming process, each root is being searched in the list of unwanted words and if that root exists in the unwanted words file, then the word will be deleted from the input file and it's derivatives should be removed from the input file as well. Finally, reconstruct the line that has unwanted words eliminated and push it in a new file (Intermediate).

3.1.2. Feature extraction:

Transforming the input data into the set of features is called feature extraction. The features have to be chosen carefully. By that, the features set will extract the relevant information from the input data in order to perform the desired task using this reduced representation instead of the full size input [13].

In features extraction, the process starts by reading the intermediate file line by line. Then, we find the stem of each word by searching in the stem file. If the stem is found then ignore the word, and go to the next word. If that stem is not found, then increase the count of features by one and add that stem into the feature set, then write that original word of that stem in the output file. We count the feature in this step in order to find the relationship between the number of features and number of example later in this research.

3.1.3. Nominal-to-numerical conversion:

To make the classification less computational expensive, the classes were numbered and their corresponding meaning was defined. Also the extracted features were transformed into numerical features usable for machine learning.

Figure 3.2 shows an example of the pre-processing phases, where un-necessary word (Handbook) is removed in the data cleaning and the remaining words will be extracted representing the features. Those features will be associated with numbers to deal with, which make it easier.

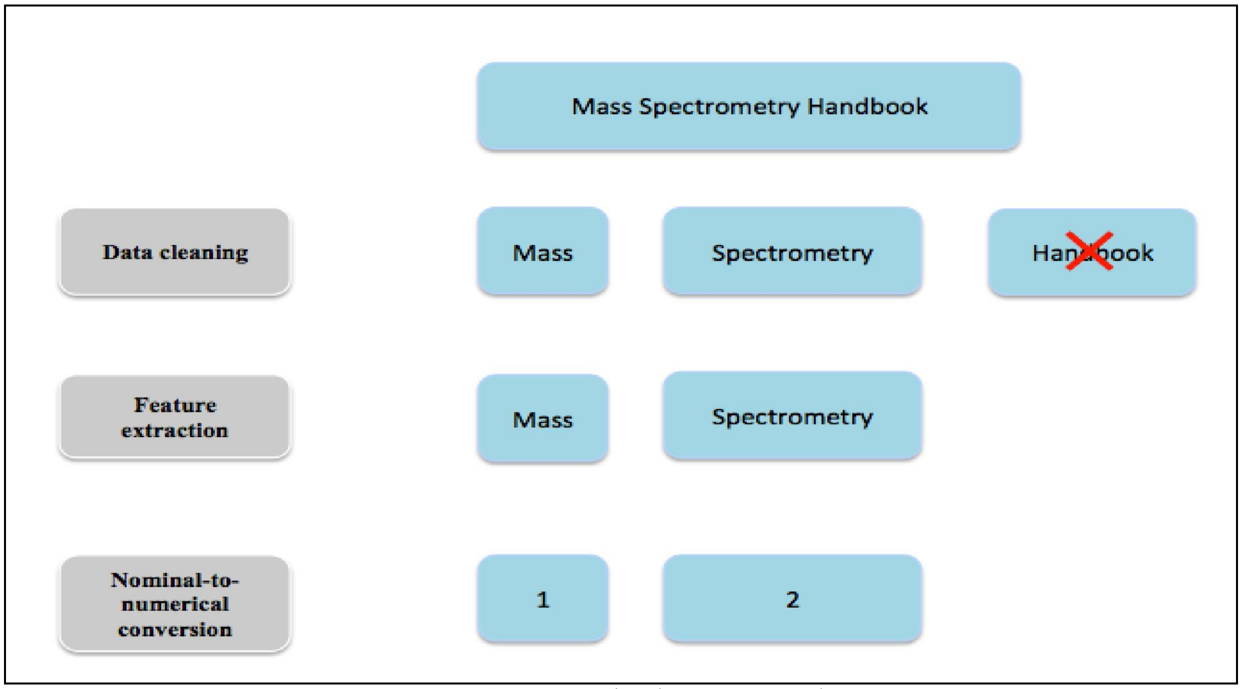

Figure 3.2: Pre-Processing phases on an example

3.2 The proposed algorithm

Once data pre-processing is completed, the data is stored in a pre-processed file to be handled later as clean data. The proposed classifiers read from this data set. First, the system reads a set of the data set X, where each example in the training $\in$ X, Then, the system reads another set of data, let’s call it Z. This data is used for testing, so each example in the testing set $\in$ Z. After the classification process is done, the error rate and the classification accuracy will be observed.

Let's call the set of misclassified examples Y, each misclassified example $\in$ Y. Then classifier must be trained again. The new training set will be called N_1, where each example in the training iteration i $\in N_i$.

Because Y is smaller than Z a number of examples (E) must be added, where E = Z – Y and the new training set is N where N = E + Y.

The system reads another set of data, let's call it (V), and this is going to be used for testing, so every example in the testing iteration i $\in V_i$.

For every classification iteration, a training session will start again and a new testing session V_i will also go through the classifier. In Figure 3.4 a simplified schema of the proposed approach is presented.

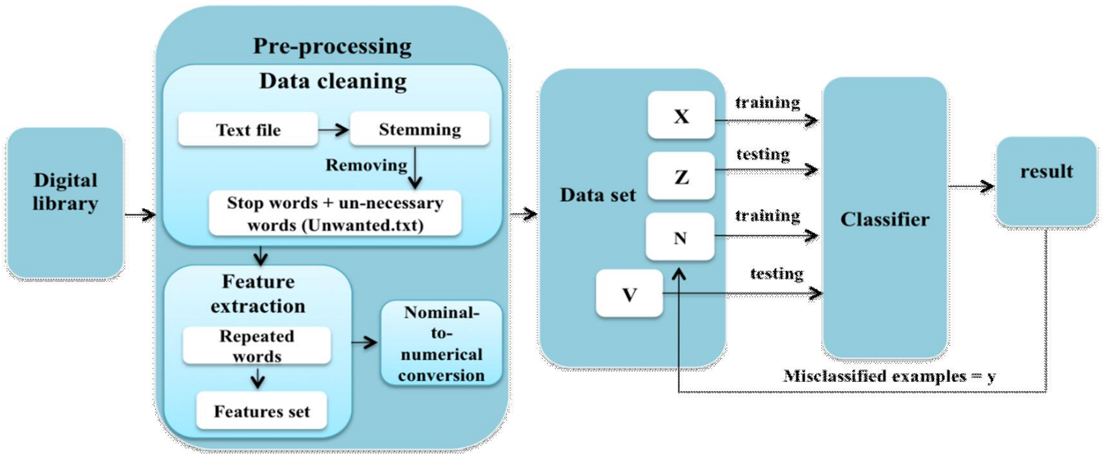

Figure 3.4: Methodology work flow

A common approach for building a reliable classifier is to split a data set in to a training set and an independent test set, where the training set is used to develop the classifier and the testing set is used to evaluate its performance. The common used strategy is allocating 2/3rd of cases for training is nearly optimal for reasonable sized data sets ($n \geq 100$) with strong signals [11]. According to this principle the workflow is as following:

- Once the data it is cleaned, the algorithm reads it.
- The algorithm trains the classifier by assigning the feature numbers with every class in the classification tree.

For example if data has a set of features (Computer = 1, Science = 2, Machine = 3, Learning = 4, Algorithm = 5, Engineering = 6, Biology = 7, Chemistry = 8), these features are assigned to each class according to a pre-designed classification tree:
Class 1, features [1, 2, 3, 5, 8, 10]
Class 2, features [1, 5, 8]
Class 3, features [5, 8, 10]
and so on.
Then, the classification tree might look as Figure 3.5.

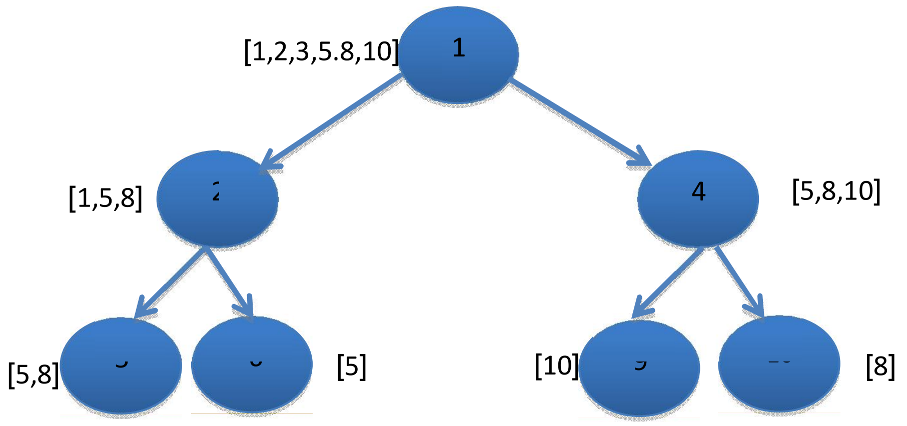

Figure 3.5: Classification tree

- Once the features are assigned to classes, the testing set is introduced, every word in the title being assigned to a class. The word might be assigned to more than one class, but only to the ones belonging to the same grandparent. Once a word cannot be classified in any class, it means, the feature of the word is new, and the classifier needs to be re-trained. The error rate is calculated if a word feature was miscounted or if a word was classified in a wrong class. Macro- and Micro- averaging are used to calculate the error rate in case a word was classified into a wrong class.

The pseudo-code of the proposed algorithm is the following:

```
TitleList = Import_Titels_List( )
CategoriesList  = Import_Categories_List( )
StemList = Import_Words_List( )
ErrorsCounter = 0
WordsCounter = 0
For each title in TitleList
    WordsIn Title = extractWordsFromTitles(title)
    WordsCounter = WordsCounter + NumberOfElement(WordsIn Title)
    TitleCategories(title) = emplylist( )
        for each wordintitle in WordsInTitle
            for each stem in StemList
                StemIsFound = false
                if StemOf (wordintiltle) = stem
                        TitleCategories(titles) = union( TitleCategories(title),
                                                        CategoriesOfStem(stem))
                    GoToNextWordIntitle( )
                    StemIsFound = true
                end
            end
            if StemIsFound == false
                    OutputWarning("The word"wordintitle "in the title"title "has
                                        not  a matching in the list of stems")
                    ErrorsCounter = ErrorsCounter + 1
            end
    end
end
ErrorRate = ErrorCounter / WordsCounter
Output("The error rate is" ErrorRate)
```

3.3. Complexity analysis

To analyze the complexity of the algorithm the following symbols are used:

n : number of titles to be analyzed
m : number of stems present in the database
t : number of categories per word (mean value)
p : number of words per title (mean value)
q : number of characters per word (mean value)
w : total number of categories

The overall number of instructions is

$$f(n,m,t,p,q) = npq + mp + n(pm(t+t+q)) + p + pq \tag{7}$$

That is in the expanded form

$$f(n,m,t,p,q) = mnpq + 2mnpt + mp + npq + pq + p \tag{8}$$

Analyzing the expression above we can note that the increasing the size of inputs the dominants terms are $mnpq$ and $2mnpt$. Then, considering that the number of categories per word (t) is generally lower than the number of characters per word (q) the time-complexity of the algorithm is $O(mnpq)$.

Considering that the number of word per title (p), the number of characters (q) and the number of categories per word (t) does not increase by increasing the input size (as they mainly depend on the language the words belong) they can be treated as constant (the medium value is considered) and can be neglected in the evaluation of the time - complexity of the algorithm.

In the end, the time complexity of the algorithm result using the big-O notation.

$$O(mn) \tag{9}$$

The space complexity of the algorithm is calculated considering the bytes of memory needed for the execution of the algorithm. Therefore, the number of bytes is defined by Equation 19, described in its extended form by Equation 20:

$$f(n,p,q,m,w) = npq + m(p+q) + pq + wq \quad (10)$$

$$f(n,p,q,m,w) = npq + mp + mq + pq + wq \quad (11)$$

Considering that the number of categories (□) is generally lower than the number of titles to be analyzed (□) and lower than the number of stems (□), in the asymptotic analysis the last term (□□) can be neglected. In these conditions, the space required when increasing n and q can be approximated as:

$$f(n,p,q,m,w) = npq + mp + mq \quad (12)$$

Considering that the number of words per title □ and the number of characters per word □ do not increase increasing the input size (as said in the previous paragraph they mainly depend on the language used), the space required can be approximated to

$$f(n,p,q,m,w) = npq + mp + mq = npq + m(p+q) \quad (13)$$

In the end, the space complexity of the algorithm result using the big-□ notation

$$O(n+m) \quad (14)$$

3.4. Case studies

The same experiment is conducted for full domain, and sub-domains of the library collections which are shown in Figure 3.6.

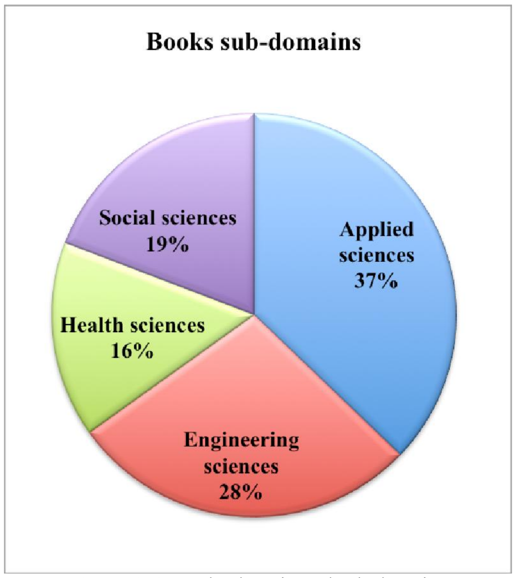

Figure 3.6: Books domain and sub-domains

3.4.1. Case study one (Willey online library)

The data set that first case study is represented by the Willey online library [70]. It has collection of books (examples) described by different attributes. These books were collected from different fields and disciplines. The characteristics of this database are the following:

- Dataset name: Wiley online library [57]
- Number of attributes: 5888
- Number of examples: 8842
- Number of classes: 64
- Number of hierarchical levels: 3

The data set already contains nominal attributes, many values were missing. Therefore, pre-processing was necessary. According to the workflow of the proposed algorithm, before training and testing, a data cleaning step and nominal to numerical conversion steps are performed.

In the data cleaning step, rare classes or classes that may have a representation of less than 1% of the data set will be ignored as 1% is really a small number of examples. In case the data set is a large one, 1% might be taken into consideration in other data sets. Some examples of books titles that might be ignored due to the low class representation are: “It Happened One Night”, “Top Hat”, “Hairspray”, “The Act of Remembering”, and “Women at the Top”.

In the nominal to numerical conversion, numbers are manually assign to each class as those classes are already induced and defined to their corresponding meaning. Also transform the extracted features into numerical features is useful for machine learning since its easy to handle when coding. For example if we have the word “science = 1” in the feature set and we got a book titled with “computer science algorithms”, the word “computer” will be assigned to 2, and the word “algorithms” will be assigned to 3 as we already have “science” assigned to 1. So, the example representation will be: “2,1,3.”

The data set has thousands of examples. To insure precise performance evaluation, a 5-fold cross validation was used. The training examples are described by thousands of attributes, thus it becomes easy to classify discriminant classes, but that means that a large number of examples is required in this case.

This data is cleaned and all un-necessary words and stop words are removed. Thus, using stemming every word is associated with its family. Consequently, words like computer, computing, computers, and compute will have only one stem number. We select $x_1, x_2 \dots . x_n \in X$, and X is a set of examples that consists of n examples for training. In the training stage, the features are manually assign with each class. This is called a class feature vector. Once this stage is achieved, the classifier is trained and becomes ready for testing.

For this case study, the scope was to test if the classifier can be trained and what would be the error rate. The dataset considered is represented by the entire book domain [70]. The entire data set including all major classes and sub-classes was used. The main classes are: Applied Science, Engineering Science, Health and Social Sciences.

The data set was divided into training and testing examples each training set having 200 examples and each testing 200 examples. For the training set, the examples are manually classified and the class label is updated with every example.

4. Results

In this section, the results of the simulations performed with the proposed algorithm and the algorithms chosen for comparison are presented and discussed. Several experiments were conducted on real world data sets from different fields including library collections [70], Routers- 21578 [71], and 20 Newsgroup [17] data sets. The final goal was to correctly classify a library collection into classes where the examples (books) are classified into classes and the classes are hierarchically ordered.

In order to compute the error rate for this dataset, the macro-micro averaging were used. When multiple class labels are to be retrieved, averaging the evaluation measures can give a view on the general results. For example, consider a binary evaluation measure B(TP,TN,FP,FN) that is calculated based on the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), and 2 labels c_1 and c_2.

Figure 4.2 shows the result of the experiment with a very noisy training, until 1000 examples are reached. At this point, the error rate starts to drop below 80%. The learning curve shows an error rate of 40% when almost 4000 examples are used. We expect that the curve will keep improving and the error rate keeps dropping as more examples are added. This experiment was conducted on the whole books domain. The number of attributes used in this experiment was 8555.

Figure 4.2 shows the result of applying the proposed algorithm, Where every class is labeled with the features that best describe the class. These all the examples were classified to all classes that may have their features as shown in Figure 3.5

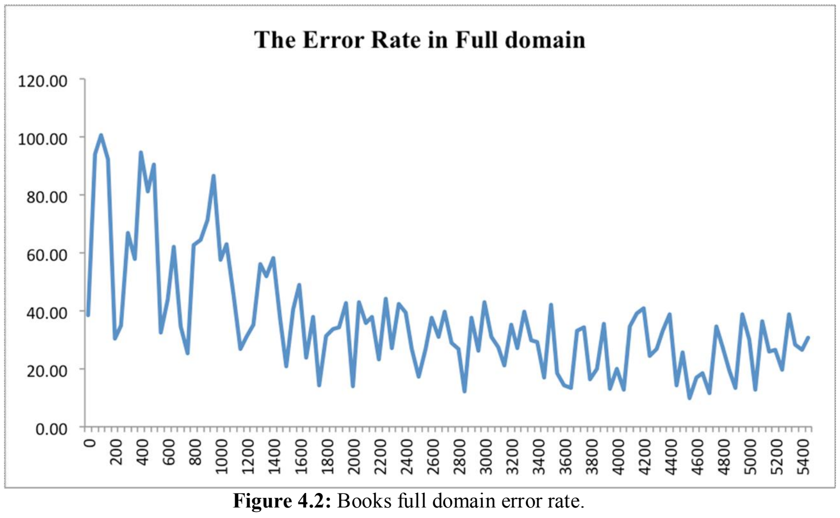

Figure 4.2: Books full domain error rate.

Figure 4.3 and 4.4 show the output of different subdomains, where the error rate drop below 20% with fewer examples. This can be explained by the fact that the number of attributes associated with this domain is very small and equals to 1831 for the engineering domain. The error rate was calculated using the Macro-Micro averaging metric.

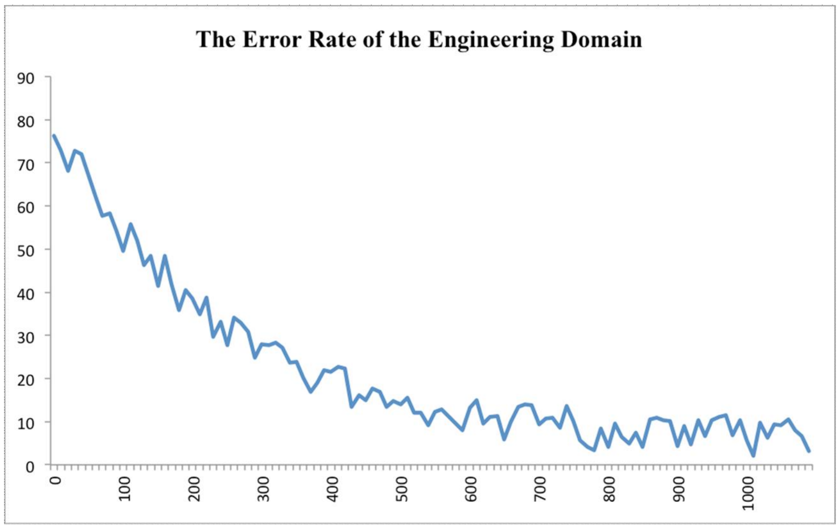

Figure 4.3: Engineering domain error rate

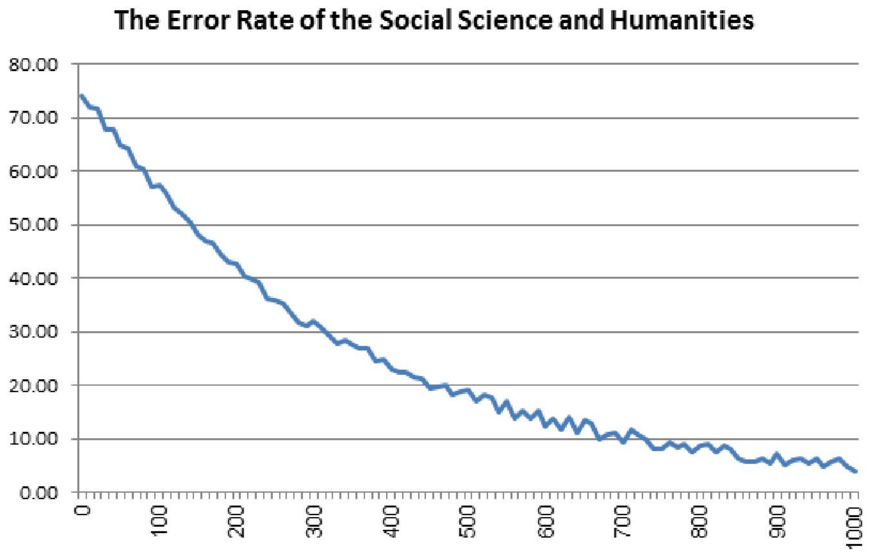

Figure 4.4: Social sciences and Humanities error rate.

The previous case studies showed that a book collection could be trained and the error rate can be reduced if more examples are added. The error rate in this case was calculated using the hamming loss [61]. Figure 4.10 shows the result of the experiment. As it can be observed, the error rate is very low starting at ≈ 1.5% to less than 0.5%.

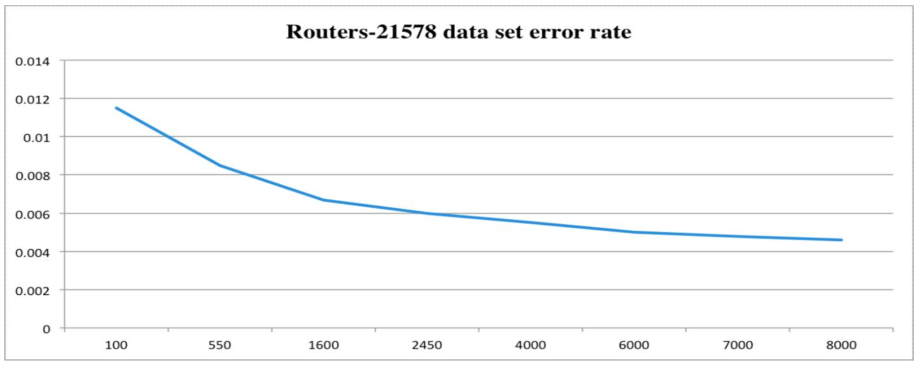

Figure 4.10: Routers-21578 data set error rate

In this experiment, we used another popular algorithm in documents classification called Term Frequency Inverse Document Frequency (TF-IDF) [8] and the data set “20 newsgroup” was used. As explained in section 3.4.8 the data set has about 19,000 documents. It is expressed in terms of the document-term matrix. Rows are represented by the document examples, and columns represent words. A matrix entry (i,j) represents the frequency of occurrence of a word j in a document i. Word frequencies for about 60,000 words are specified for each document. The item (1,1) means document #1, word #1, and the #4 means word#1 has frequency = 4 in document #1 and so on.

In an attempt to reduce the dimensionality of dataset, the following steps were preformed:

- Removing features that do not help in discriminating between class i.e., words like ’a’, ‘the’ that appear in all documents.
- Using Principle Component Analysis PCA [39] for dimensionality reduction

Words with high Inverse Document Frequency (IDF) counts are removed, where IDF represents the ratio of the number of documents in which a particular word appears, to the total number of documents. A high value indicates that the word is present in most of the documents across classes, and hence does not help much in discriminating between the classes. But we noticed that we were left with a large number of words even after removing such words with counts above a certain threshold. Since discarding information can affect classifier performance later, setting too low threshold is not a good thing we dropped this and looked at alternative attempts.

4.1. Existing algorithm (Method A: Naïve Bays algorithm)

The first task in the training stage is to separate 10% data for testing purposes from each class. 10% data is separated for each class out of total data for that class. For example, if 100 samples are available for class 1, 10 samples were taken out for testing. This 10% amount is standard in literature and in normal circumstances 10-15% data is taken out for testing. If 50% of data is taken out, too less remains for the training stage and the classifier may not generalize well. After segregating, there were 9012 items in training data and 969 items in testing data, including roughly 10% from each class.

On the second stage, the methodology of data cleaning presented in section 3.1is applied, all the unnecessary words being removed from the titles of the books. These include articles such as, (a, and), prepositions (of, for etc.) and other common meaningless words like volume, edition, e-book etc.

On the third stage, all the remaining words were extracted from the books’ titles and each word was assigned a unique number and another identifier to show which class it belongs to. Actually wordID is unnecessary and we may ignore it. For each word, we have a list of classes it can belong to, e.g. chemical may belong to class 1, 3 and 5 so its class ID will be (1,3,5). So the format was like:
word class ID wordID

For instance, the word Horticulture, the word ID may be 5 and if it belongs to class 10, its class ID is 10. Then all words are converted to uppercase so that while comparing the words later we do not have to deal with case-sensitivity issue.

Finally, all the words are sorted alphabetically so that when comparing in the testing stage, we do not have to compare with all words. Thus, there is no need to compute distances with all the words but only with those which start with the specified letter. For instance, if the word is Horticulture, then we only need to compare it with the words starting with the letter H.

Once all is done, all the words, sorted alphabetically and in the format mentioned above, are stored in a data file.

4.1.1. Testing stage

In the testing stage, all the titles, along with the class to which it belongs to are passed through the testing function. For instance:

Class	title
Agriculture	Horticultural Reviews, Volume 1

In the testing stage, the book title goes through the same steps as the training data start the data cleaning process again. Useless words are removed, but useful words are extracted and separated, then converted into uppercase.

After pre-processing, the final shape of the title will be:
(HORTICULTURAL REVIEWS and VOLUME and 1) all of those words being removed as they are very generic words and cannot be associated with a particular class. The only word kept is HORTICULTURAL

Next, the training data is loaded and as discussed before, only the words starting with the same letter as the testing word are selected for comparison. For instance, when we want to see which class the word FUNGI belong to, we will compare only with the words starting with the letter F. We do the same with the example above HORTICULTURAL which starts with H, we will compare it only with the words starting with the letter H. We call this a stemming process, where we associate each word with its own stem words only.

The testing word is compared with the above selected words and its distance is computed from them. Two types of matching techniques are used; one is the Lavenstein Distance [5], the number of edits required to convert one string to another. For instance, if HORTICULTURE and HORTICULTURAL are compared, then Lavenstein Distance will be 2, since 2 edits are required to convert the first string into the second one.

4.1.2. Using single label classification

The testing is done for all the classes and the error rate for each class is compared. The error rate for most of the classes is quite high, the reason being that out of the 49000 total words in the 9012 training titles, only about 8500 are unique and the rest are just repetitions. This indicates a huge overlap of data among different classes and as a consequence, the classifier gets confused while testing and therefore misclassify the data.

The last step, in which we pick the ID of the most frequent class, normally has number of IDs with the same frequency, and just picking the first one also introduces errors.

The average error rate for all the classes can be computed by

$$\text{Avg. Error Rate} = \frac{\sum \text{(Error Rate of class) x (No. Words in the class)}}{\text{Total No. of Words}}$$

where ∑ indicates the sum over all the classes. This is just the concept of weighted average. e.g. if we have 3 classes; class 1 has an error rate of 60%, class 2 has an error rate of 40% and class 3 has an error rate of 50%.

Then using the simple average formula, the error rate obtained is 50%. ((60+40+50)/3 = 50). But now suppose there are total of 10 words; 5 belong to class 1, 3 to class 2 and 2 to class 3. Since more words belong to class, logically its error rate should have more contribution in the overall error rate. So we do weighted average, weight for class 1 is 5/10 = 0.5 (no. of words in the class / total words). Similarly weight for class 2 is 3/10 = 0.3 and weight for class 3 is 2/10 = 0.2. Now we multiply with respective error rates and sum them up; so the error rate becomes (0.5 x 60) + (0.3 x 40) + (0.2 x 50) = 30 + 12 +10 = 52%. This error rate is more indicative of the overall behavior of all the classes as it gives more weight to the classes with more amount of data. Using the above formula, the Avg. Error Rate for all classes comes out to be 64%.

4.1.3. Using multi-label classification

The error rate can be reduced by number of different techniques. One simple way is to get all the different class IDs in the last step of testing stage, each of which have the same probability to be assigned to the given title – this is the Multi-Label Classification and in this case each title can belong to multiple classes. Given that the actual class is among the final set, this step can eliminate all the non-probable classes and another classifier can be used in the next step to choose the final class or a human can do that provided the number of such instances are small.

Alternatively, we can extract the unique words and find out for each word the class to which it belongs most frequently – the bag-of-words approach, and then instead of assigning the class IDs of all the classes to which the word may belong, we assign only those class (or classes) IDs to which it belongs the most. But this technique is biased towards the class having more training samples, and the error rates for the classes which have the lesser data may increase more. But the overall error rate will surely decrease as the classifier is now more biased towards the classes which are more frequent and more likely to come.

The Avg. Error Rate for this technique comes out to be 57% which is an improvement over the previous technique.

4.2. Existing algorithm (Method B: KNN algorithm)

Suppose the classifier was asked to classify some sample X, and after computation classifier finds that it can belong to any one of the class 1, class 2 and class 3. In this case, we need a rule to break a tie and the one we used in previous section was to pick the lowest class ID. Therefore, the sample will be classified to class 1 even though it may belong to class 2 or class 3. Suppose that sample originally belonged to class 2; then the classification will be wrong and will account as an error. But if we don't use any tie-breaker and outputs all the equally probable classes, i.e. class 1, class 2 and class 3, then there will be no error as sample does belong to one of these classes. This is the whole idea of multi-label classification or multi-output classification, in which input X is not mapped to a single scalar class y, but rather a vector of classes Y

The algorithm was modified to accommodate the multi-label classification and an error occurs only if the actual class y was not among the vector of classes Y given by the classifier. The error rate is reduced in this case compared to the case of single-output classification.

Green blocks indicate the classes with error rate of less than 50% and it is visible that now such instances are lot more than in previous tables. The overall Avg. Error Rate is 41% which is an improvement of nearly 16% from the previous method.

4.2.1. Classification of parent class

The classification of Parent Classes is much less error prone as there are few parent classes and the margin for error is smaller. In this case, there are 6 Parent classes corresponding to 65 Child classes. The original classifier (in multi-label case) returns a set of child classes, which may or may not belong to the same parent class. So we get the parent class for each of these child classes and then compare them one by one to the original parent class. An error occurs only if none of the parent classes match the original parent class.

The highlighted entries indicate the classes with error rate of less than 30%. The overall weighted Average Error Rate is 21%.

5. Conclusion

This study is focused on the HMC with emphasis on several case studies to draw the research observations. This is done by conducting various experiments including many popular machine learning algorithms. KNN and Naïve Bays algorithms along with the proposed algorithm based on SVM are used. The research also aimed to identify the child-parent relation, and parent-child relation. To this goal, a proprietary software was built to test whether an example is classified into its corresponding child and grandchild as well as if the grandchild belonged to its accurate parent and grandparent. The significance of the research is the motivation for the use of machine learning in digital libraries which were the primary resource that were used in the study. We have also used 20 newsgroup and Routers data set to compare the performance.

The performance analysis was done using Macro and Micro averaging and hamming loss metrics. Based on the results, it was found that, it is very time consuming and costly to use only book titles to classify a large collection of library contents. Thus, we suggest that we either use the sub domain approach or classify each sub domain separately, or include more information such as abstract of documents, an introduction of the document. Another major finding is that a parent node can be a parent of all documents with a small and acceptable error rate. In general, our findings are very similar to the many recent published studies.

In future, we plan to generalize the proposed algorithm for the hierarchical case where the interrelation of the class labels can be specified by a generalization tree of a directed acyclic graph (DAG) as in Vateekul et al. [2] study.

References

1. E. Alpaydin, Introduction to machine learning: MIT press, 2004.
2. P. Vateekul, M. Kubat, and K. Sarinnapakorn, "Hierarchical Multi-Label Classification with SVMs: a Case Study in Gene Function Prediction." . unpublished work.
3. C. Vens, J. Struyf, L. Schietgat, S. Dzeroski, and H. Blockeel, "Decision trees for hierarchical multi-label classification." Machine Learning, vol. 73, no. 2. pp.185–214, 2008.
4. T. Joachims, Text categorization with support vector machines: Learning with many relevant features: Springer, 1998.
5. V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals." Soviet physics doklady vol. 10, p.707. 1966
6. L. Baoli, L. Qin, and Y. Shiwen, "An adaptive k-nearest neighbor text categorization strategy," ACM Transactions on Asian Language Information Processing (TALIP), vol. 3, no. 4. pp.215-226, 2004.
7. N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," Machine learning, vol. 29, no. 2-3. pp.131-163, 1997.
8. A. Clare and R. D. King, "Predicting gene function in Saccharomyces cerevisiae," Bioinformatics, vol. 19, no. suppl 2. pp.ii42-ii49, 2003.
9. A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification." AAAI-98 workshop on learning for text categorization vol. 752, pp. 41-48. 1998. Citeseer.
10. J. T.-Y. Kwok, "Automated text categorization using support vector machine." In Proceedings of the International Conference on Neural Information Processing (ICONIP) . 1998. Citeseer.
11. K. K. Dobbin and R. M. Simon, "Optimally splitting cases for training and testing high dimensional classifiers," BMC medical genomics, vol. 4, no. 1. pp.31, 2011.
12. L. Schietgat, C. Vens, J. Struyf, H. Blockeel, D. Kocev, and S. Dzeroski, "Predicting gene function using hierarchical multi-label decision tree ensembles." Bmc Bioinformatics, vol. 11, 2010.
13. K. J. Cios, R. W. Swiniarski, W. Pedrycz et al., "Feature Extraction and Selection Methods." Data Mining. pp. 133-233. 2007. Springer.
14. D. Koller and M. Sahami, "Hierarchically classifying documents using very few words,", 1997.
15. M. F. Porter, "An algorithm for suffix stripping," Program: electronic library and information systems, vol. 14, no. 3. pp.130-137, 1980.
16. W. B. Frakes and R. Baeza-Yates, "Information retrieval: data structures and algorithms,", 1992.
17. "The 20 Newsgroups data set." http://qwone.com/~jason/20Newsgroups/ . 2008.
29. C. N. Silla Jr and A. A. Freitas, "A survey of hierarchical classification across different

application domains," Data Mining and Knowledge Discovery, vol. 22, no. 1-2. pp.31-72, 2011.
33. A. Sun and E. P. Lim, "Hierarchical text classification and evaluation." Data Mining, 2001.ICDM 2001, Proceedings IEEE International Conference on , pp. 521-528. 2001. IEEE.
34. C. D. Nguyen, T. A. Dung, and T. H. Cao, "Text classification for DAG-structured categories." Advances in Knowledge Discovery and Data Mining. pp. 290-300. 2005. Springer.
35. A. Secker, M. N. Davies, A. A. Freitas et al., "An experimental comparison of classification algorithms for hierarchical prediction of protein function," Expert Update (Magazine of the British Computer Society's Specialist Group on AI), vol. 9, no. 3. pp.17-22, 2007.
36. W. Bi and J. T. Kwok, "Multi-label classification on tree-and dag-structured hierarchies." Proceedings of the 28th International Conference on Machine Learning (ICML-11) , pp. 17-24. 2011.
37. N. Alaydie, C. K. Reddy, and F. Fotouhi, "Exploiting label dependency for hierarchical multi-label classification." Advances in Knowledge Discovery and Data Mining. pp. 294-305. 2012. Springer.
38. J. R. Quinlan, "Induction of decision trees," Machine learning, vol. 1, no. 1. pp.81-106, 1986.
39. "Dimensionality reduction." http://en.wikipedia.org/wiki/ Dimensionality_reduction, 2014.
40. H. Blockeel, L. De Raedt, and J. Ramon, "Top-down induction of clustering trees," arXiv preprint cs/0011032, 2000.
41. G. Pandey, C. L. Myers, and V. Kumar, "Incorporating functional inter-relationships into protein function prediction algorithms," BMC bioinformatics, vol. 10, no. 1. pp.142, 2009.
42. H. Lo, S. Lin, and H. M. Wang, "Generalized k-Labelsets Ensemble for Multi-Label and Cost-Sensitive Classification,", 2013.
43. G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data." Data mining and knowledge discovery handbook. pp. 667-685. 2010. Springer.
44. G. Tsoumakas and I. Vlahavas, "Random k-labelsets: An ensemble method for multi label classification." Machine Learning: ECML 2007. pp. 406-417. 2007. Springer.
45. "Mulan: A Java Library for Multi-Label Learning." . 2014.
46. H. Qu, S. Zhang, H. Liu et al., "A multi-label classification algorithm based on label-specific features," Wuhan University Journal of Natural Sciences, vol. 16, no. 6. pp.520-524, 2011.
47. X. Kong, B. Cao, and P. S. Yu, "Multi-label classification by mining label and instance correlations from heterogeneous information networks." Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining , pp. 614-622. 2013. ACM.
48. B. Chen, Y. Ding, and D. J. Wild, "Assessing drug target association using semantic linked data," PLoS computational biology, vol. 8, no. 7. pp.e1002574, 2012.
49. X. Kong, P. S. Yu, Y. Ding et al., "Meta path-based collective classification in heterogeneous information networks." Proceedings of the 21st ACM international conference on Information and knowledge management , pp. 1567-1571. 2012. ACM.
50. X. Kong, X. Shi, and S. Y. Philip, "Multi-Label Collective Classification." SDM vol. 11, pp. 618-629. 2011. SIAM.
52. L. Enrique Sucar, C. Bielza, E. F. Morales et al., "Multi-label classification with Bayesian network-based chain classifiers," Pattern Recognition Letters, 2013.
57. A. Santos, A. Canuto, and A. F. Neto, "A comparative analysis of classification methods to multi-label tasks in different application domains," Int.J.Comput.Inform.Syst.Indust.Manag.Appl, vol. 3. pp.218-227, 2011.
58. M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: one-sided selection." ICML vol. 97, pp. 179-186. 1997.
59. A. Singhal, "Modern information retrieval: A brief overview," IEEE Data Eng.Bull., vol. 24, no. 4. pp.35-43, 2001.
60. Y. Yang, "An evaluation of statistical approaches to text categorization," Information retrieval, vol. 1, no. 1-2. pp.69-90, 1999.
61. V. Gjorgjioski, D. Kocev, and S. D++eroski, "COMPARISON OF DISTANCES FOR MULTI-LABEL CLASSIFICATION WITH PCTs,"
65. D. Damm, C. Fremerey, V. Thomas et al., "A digital library framework for heterogeneous music collections: from document acquisition to cross-modal interaction," International Journal on Digital Libraries, vol. 12, no. 2-3. pp.53-71, 2012.
66. "Internet Archieve." http://archive.org/index.php . 2014.
67. "Google Books." http://books.google.com . 2014.
68. "Open Library." http://openlibrary.org . 2014.
69. "New York Public Library." http://www.nypl.org . 2014.
70. "Wiley Online Library." http://onlinelibrary.wiley.com/ . 2014.
71. D. D. Lewis, "Reuters-21578 text categorization test collection." . 2014.