A Study of multilabel text classification and the effect of label hierarchy

Sushobhan Nayak[1], *Raghav Ramesh*[2], *Suril Shah*[3]

CS224N Project Report,
Stanford University

nayaks@stanford.edu, raghavr@stanford.edu, surils@stanford.edu

1. Introduction

Text classification has traditionally been one of the most popular problems in information retreival, natural language processing and machine learning. In the simplest case, the task of text classification [1] is as follows: A set of training documents $T = \{X_1, X_2, ...X_m\}$, each labelled with a class value from a set of k distinct labels, from the set $\{1, 2, ..k\}$, is used to learn a classification model, that captures the relationship between features in the documents and their labels. Subsequently, for a test document with unknown label, the model is used to predict a label. The problem of text classification finds wide appeal in various domains for tasks such as (i) News selection and grouping, (ii) Document organization in digital libraries, websites, social feeds,etc., (iii) Email classification including spam filtering. A variation of this problem in which each document can belong to any number of classes (labels) is referred to as a multilabel text classification problem. An extension to such a problem in which the labels are interrelated by a categorical hierarchy is referred to as a hierarchical text classification problem. In this project, we

- study the task of multilabel text classification on real world datasets
- implement algorithms to leverage the hierarchy in the labels
- analyze the effect of different algorithmic approaches and dataset properties on the classification performance

2. Background

Multilabel classification methods find applications[2] in many fields including protein function classification, music categorization [5] and semantic scene classification[6]. A classic case of multilabel classification is text categorization of news articles, for example, a news article on Obamacare is labelled with both ”Politics” and ”Healthcare”. The labels in a dataset can belong to a hierarchical structure [3], for example, labels can be organized as a tree in tasks in functional genomics[4] or a DAG in Gene Ontology [3].

Multilabel classification methods are primarily grouped under two categories [2] -(i) problem transformation methods and (ii) algorithm adaptation methods. Problem transformation methods transform the multilabel task into either one or more single label classification or regression tasks. Algorithm adaptation techniques extend specific algorithms to handle multilabel data directly.

2.1. Problem transformation methods

Naive transformation methods include (i) random selection of one label for each multilabel instance or (ii) removal of multilabel instances from the dataset. These methods are clearly pretty naive as they discard a lot of information. Another transformation technique is to consider each label subgroup occuring in the training set as a single label. Thus, it learns one single label classifier for every element (label) of the power set of labels. The drawback of this method is it leads to a large number of classes and only a few examples per class. Another transformation technique is to adopt the approach used in single label multiclass classification using a binary classifier, where one binary classifier is learnt for each label by transforming the original dataset D into $|L|$ datasets (L being the set of all labels), where each dataset D_i contains all examples of the original dataset with the label i if in D, the example contained the label i^1 and i^0 otherwise.

2.2. Algorithm Adaptation methods

There exists rich literature on adaptation of algorithms for multilabel task. To give a brief idea, we explain one such method, ML-kNN [7] which is an adaptation of kNN algorithm. In short, ML-kNN uses the kNN algorithm independently for each label *l* and finds the k nearest examples to the test instance and considers those labelled with *l* as positive and the rest as negative. The main differentiation in this method from a direct application of the original kNN algorithm using the problem transformation method described above is the use of prior probabilities. Similar adaptation methods have been proposed for other algorithms including SVM.

2.3. Hierarchical Classification

In many applications of text categorization, the labels are often organized in a tree-structured hierarchy. An instance is associated with a certain label only if it is also associated with the labels parent in the hierarchy. However, most existing multi-label classification algorithms do not take the label structure into consideration. Instead, the labels are simply treated separately, leading to the need to train a large number of classifiers (one for each label). Further, as some leaf labels may have very few positive examples, the training data become highly skewed, creating problems in many classifiers. Besides, the inconsistent labellings between child and parent causes difficulty in interpretation. Finally, the prediction performance is impaired as structural dependencies among labels are not utilized in the learning process. Some recent approaches do the following: (citations from cssag paper) predict positive for a label only if the parent is predicted; construct train examples for each node from samples of the parent node; use large-margin structured predictors, or by adapting decision trees.

3. Problem Formulation

3.1. Algorithms and Datasets

The gist of the project is to do a study of how different classification algorithms perform on different datasets. Towards that end, we judiciously chose algorithms that cover the spectrum we have described in the previous section, and chose datasets such that they varied in different dimensions such as average length, total vocabulary, average frequency of frequent words, sentential structure, sheer amount and the ratio of labels to documents and features. We chose 4 datasets that reflect these properties, and which we touch upon in the next section. In the algorithms front, our experiment involved standard naive-bayes, SVM, kNN, random forests to incorporation of state-of-the-art neural network based models like BPMLL and unsupervised RBMs. We also implemented a deep neural net for text classification, with 3 hidden RBM layers. The unsupervised RBM was essentially used to transform the huge vocabulary space into a lower dimensional space. Thereafter, all the algorithms are run on the transformed feature space. So, effectively, we ran around 15 classifiers across 4 datasets, and we also look into different aspects of the individual classifiers for each dataset, particularly for the hierarchical learning algorithm, which along with the implementation gives us a massive result set to derive our conclusions upon.

3.2. Evaluation

In multi-label classification, the prediction may be right, wrong or partially right, because, in the case documents belong to two or more classes, the projection can hit them all (classes), any of them or just a few of them. Several measures have been proposed to evaluate the multi-label classifiers [9]. They are divided into example-based and label-based evaluation measures [10]. The first are calculated based on the average differences of the actual and projected set of labels over all test examples. The later decomposes the evaluation process into separate evaluations for each label, and then calculates the average of all labels. Hamming Loss and Classification Accuracy [11] are example-based evaluation measures. Binary evaluation measures, such as accuracy, precision and recall, can be used as a label-based evaluation measure. We will focus on these later measures as they are widely used and they are conceptually closer to measures we use in multiclass single label scenarios. Traditionally, these measures are averaged in two ways: *micro* and *macro*. Let TP_t , FP_t , FN_t denote the true-positives, false-positives and false-negatives for the class-label $t \in T$. The micro-averaged F1 is:

$$P = \frac{\sum t \in TTP_t}{\sum_{t \in T} TP_t + FP_t}$$

$$R = \frac{\sum t \in TTP_t}{\sum_{t \in T} TP_t + FN_t}$$

$$F1_{micro} = \frac{2PR}{P + R}$$

Macro-F1, unlike Micro-F1 which gives equal weights to all instances in the averaging proccess, is partial to giving uniform weight to class-labels.

$$P_t = \frac{TP_t}{TP_t + FP_t}$$

$$R_t = \frac{TP_t}{TP_t + FN_t}$$

$$F1_{macro} = \frac{1}{|T|} \sum_{t \in T} \frac{2P_t R_t}{P_t + R_t}$$

Since both the measures give importance to different factors in a classification, usually in preactice we look at both to get an idea of how the algorithm is performing in different fronts.

Notice that, these metrics still don't take care of cases when we are learning on a hierarchy. In such cases, even if two predicted labels are not the same, if they are close together in the hierarchy tree, then in principle they should be getting a favorable weight, which notion is absent from the above formulation. There have been alternative formulations for this , but we would not consider them here primarily because we don't have too much precedence to compare them with and secondly because lots of hierarchical algorithm papers also focus on just micro- and macro-averaged F1 scores. Also, using the same metric for both gives us a way to compare them on the same footing to see if incorporation of hierarchical knowledge actually leads to any improvement in performance.

4. Dataset Description

Since we are interested in evaluating the strengths and weaknesses of a multitude of algorithms in a multi-label text classification context, we decided to use datasets with diverse characteristics. We obtained datasets that have a very high number of features, and a relatively low number of labels (Citeseer and Reuters), with a feature to label ratio over 100. Conversely, we have a dataset (Delicious) with feature to label ratio under 0.5. We also have datasets with varying cardinality, from cardinality ranging from less than 2 to greater than 20. Cardinality is the number of labels per each instance. 2 of our datasets (Reuters and Citeseer) also have well-defined hierarchy, since we also want to improve upon some algorithms by exploiting the hierarchy among labels. In Table 1, we note these various properties of each dataset we used.

5. Algorithms and Implementation

5.1. Algorithms

We looked at a bunch of algorithms for multilabel classification. We describe them below:

Name	Features	Labels	Training	Test	Cardinality
Bookmarks	2150	208	20000	10000	2.028
Citeseer	19912	83	6798	2265	1.26
Delicious	500	983	12910	3181	19.02
Reuters	48734	101	23149	10000	3.18

Table 1: Datasets and their properties

5.1.1. Naive-Bayes

We use naive-bayes as a base line. Naive bayes is known to work fairly well in text classification and we wanted to see how other algorithms perform with respect to it. Our feature space consists of tf-idf weighted monogram vectors. So we use the multinomial event model for naive-bayes, which is traditionally known to give very good performances , and gives us a robust and challenging baseline. Specifically, the distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn}$ for each class y, where n is the number of features (in our present text classification problem, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y. The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{|T|} N_{yi}$ is the total count of all features for class y. We employ Laplace smoothing to account for features not present in the learning samples and prevent zero probabilities in further computations, in which case $\alpha = 1$.

5.1.2. k-Nearest Neighbor

We use the multilabel lazy ML-kNN version of nearest neighbors. In detail, for each unseen instance, its k nearest neighbors in the training set are firstly identified. After that, based on statistical information gained from the label sets of these neighboring instances, i.e. the number of neighboring instances belonging to each possible class, maximum a posteriori (MAP) principle is utilized to determine the label set for the unseen instance.

5.1.3. Random Forests

Random forests are based on randomized decision trees. Decision Trees (DTs) are a non-parametric supervised learning method which create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features, primarily leveraging information theoretic measures. The random forest uses perturb-and-combine techniques specifically designed for trees. This means a diverse set of classifiers is created by introducing randomness in the classifier construction. The prediction of the ensemble is given as the averaged prediction of the individual classifiers. In particular, each tree in the forest is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

5.1.4. SVM

Using SVM is a no-brainer, given its popularity and accuracy in most of the classification task. We use a one-vs-all version of SVM for multiclass classification. We experiment with both linear and rbf kernels.

5.1.5. RBM

Given the recent surge in neural network based models, we also wanted to explore how minor versions of the networks would work on text classification problems. Specifically, we use Random Boltzmann Machines to reduce our feature space and capture the non-linearity. Restricted Boltzmann machines (RBM) are unsupervised nonlinear feature learners based on a probabilistic model. The features extracted by an RBM or a hierarchy of RBMs often give good results when fed into a linear classifier such as a linear SVM or a perceptron. The graphical model of an RBM is a fully-connected bipartite graph.The nodes are random variables whose states depend on the state of the other nodes they are connected to. The model is therefore parameterized by the weights of the connections, as well as one intercept (bias) term for each visible and hidden unit. An RBM with n hidden units is a parametric model of the joint distribution between a layer of hidden variables (referred to as neurons or features) $h = (h_1, \ldots, h_n)$ and the observed variables made of $x = (x_1, \ldots, x_d)$ and y, that takes the form

$$p(y, x, h) \propto \exp(-E(y, x, h))$$

where

$$E(y, x, h) = -h^T W x - b^T x - c^T h - d^T \hat{y} - h^T U \hat{y}$$

with parameters $mathbb\theta = (W, b, c, d, U)$ and $\hat{y} = (1_{y=i})_{i=1}^{C}$ for C classes. We unsupervisedly the uppermost layer essentially acts as the representation of the input feature vector in the new dimensions that incorporated important non-linear information in the data.

5.1.6. Neural net

While unsupervised RBMs have been a fairly recent development, we also tried a traditional supervised neural net classifier, BPMLL [8] . This neural network structure has traditional fully connected visible and hidden layers. The novelty lies in trying to minimize a suitable error metric. For multilabel classification in particular, BPMLL uses the following error metric:

$$E = \sum_i E_i = \sum_i \frac{1}{|Y_i||\hat{Y_i}|} \sum_{(k,l)\in Y_i \times \hat{Y_i}} \exp(-(c_k^i - c_l^i))$$

where, E_i is the error of the network on data-point x_i, $c_j^i = c_j(x_i)$ is the actual output of the network on x_i on the j-th class and Y_i is the true label set for x_i and $\hat{Y_i}$ is the complementary set of Y_i. This formulation of error takes into account the correlations between the different labels of x_i, e.g. labels in Y_i should be ranked higher than those not in Y_i.

5.1.7. Hierarchical Learning

A large part of our project focuses on the effect of hierarchy on multilabel classification. Most of the algorithms described above treat the label space as flat, while in certain problems the labels could be thematically thought to be represented in a hierarchy. The previous algorithms do not leverage that property if it exists, and we wanted to see if leveraging that information can lead to better classification performance. So we would describe an algorithm due to [3] in detail, which we implemented from scratch and ran multiple experiments on.

The algorithm first processes the given inputs. It first uses the kernel dependency estimation (KDE) approach to reduce the possibly large number of labels to a manageable number of single-label learning problems. Specifically, it transforms the label space to a lower dimensional space; note that it's different than the traditional dimension reduction techniques used in classification task in the sense that it reduces the label space instead of the feature space as is usually done. Specifically, each label vector y_i is projected to a low-dimensional vector $z_i \in \mathcal{R}^m$. Using $\{(x_i, z_i)\}_{i=1}^{n}$, learn a mapping from $\mathcal{X}$ to each projected dimension $j = 1, \ldots, m$. Since the projected directions are orthogonal, these m models can be trained independently. For a test sample x, first obtain the prediction $\hat{z} = [\hat{z}_1, \ldots, \hat{z}_m]^T$ from the m learned models. This is then mapped back (decoded) to $\hat{y} \in \mathcal{R}^d$ in the original label space and followed by further rounding to $\{0, 1\}^d$. The projection step can be implemented in various ways. The original paper follows a kernel PCA approach to encode the hierarchy information in the transformation. Specifically, let N be the number of nodes in the tree. The feature vector for node i is $l(i) = [l_1, \ldots, l_N]^T \in \{0, 1\}^N$, where $l_k = 1$ if k is an ancestor of i or $k = i$; and 0 otherwise. The label kernel evaluation for two nodes i and j is then $l(i)^T l(j)$. We also do experiments with PCA to get a trade-off between too much hierarchy information and simple dimension reduction, which is a novelty in this work and sometimes gives better results than the original algorithm, a detailed analysis of which we will provide later.

After learning and prediction, the algorithm runs a post processing phase which learns labels on the hierarchy tree. The final prediction can be represented as an optimization problem of finding out the assignments with highest weight which sum up to L (if we are predicting L labels), based on certain conditions (for example, if prediction on a hierarchy, we ensure that if a child is predicted, both(AND tree) or either(OR tree) of its parents are predicted), which essentially boils down to a fractional knapsack problem that is solved with a greedy approach. Please look at the original paper for further details.

5.2. Implementation

We used readily available Python modules from sklearn for naive-bayes, RBM, and SVMs. We used the Java implementation of Mulan for MLkNN and BPMLL algorithms. The hierarchical algorithm was implemented from scratch in Python. The code is publicly available at *https://github.com/sushobhannayak/cssag*. We also used Theano to implement some Deep learning architectures for text classification, the bulk of which we submit, and even though we didn't do much with it for this project due to time constraints and the already massive amount of work, we propose to look into in a future endeavor, about which we will be shortly discussing in future works.

6. Results

We implemented all algorithms described above and tested the performance on all datasets. For each run, we obtain a probability for each label, that gives the likelihood of the instance to contain the label. There are two methods to translate these probabilities to set of labels -(i) Use a threshold value for probability and chose all labels whose probabilities exceed the threshold, (ii) Choose a particular number of labels based on sorted probabilities. Both the methods are equivalent and while we calculated error metrics from both methods, we present only the values from the first method here. Table 2 gives the precision, recall and F1 scores using micro averaging, while Table 3 gives the corresponding values under macro averaging. The values for ML-KNN and BPML for datasets Citeseer and Delicious are not given here as the Mulan implementation of these algorithms was not scalable to suit the size of the datasets.

Algorithm	Reuters	Citeseer	Bookmarks	Delicious	
	0.280	0.04	0.100	0.143	P
Naive Bayes	0.310	0.003	0.271	0.441	R
	0.294	0.005	0.146	0.215	F1
	0.404	0.515	0.426	0.405	P
SVM Kernel	0.419	0.383	0.212	0.279	R
	0.411	0.446	0.283	0.330	F1
	0.328	0.422	0.406	0.439	P
SVM rbf	0.356	0.349	0.274	0.278	R
	0.342	0.382	0.327	0.340	F1
	0.331	0.266	0.369	0.397	P
Random Forests	0.406	0.450	0.241	0.351	R
	0.365	0.335	0.292	0372	F1
	0.349	0.300	0.369	0.356	P
Extra Trees	0.395	0.458	0.248	0.356	R
	0.371	0.362	0.297	0.356	F1
	NA	NA	0.848	0.651	P
MLkNN	NA	NA	0.132	0.101	R
	NA	NA	0.228	0.175	F1
	NA	NA	0.010	0.118	P
BPML	NA	NA	0.919	0.727	R
	NA	NA	0.020	0.203	F1

Table 2: Micro Precision, Recall and F1 of flat classification algorithms

Figure 1 shows the F1 scores obtained under different algorithms for each dataset. As we see from the figure, values obtained for Citeseer and Reuters dataset are higher than those for the other datasets. The properties of the text in these datasets that potentially causes this

Algorithm	Reuters	Citeseer	Bookmarks	Delicious	
	0.147	0.037	0.123	0.069	P
Naive Bayes	0.121	0.003	0.229	0.262	R
	0.097	0.005	0.125	0.102	F1
	0.378	0.486	0.220	0.159	P
SVM Kernel	0.335	0.421	0.137	0.133	R
	0.315	0.437	0.147	0.105	F1
	0.358	0.452	0.277	0.202	P
SVM rbf	0.307	0.372	0.201	0.144	R
	0.263	0.352	0.216	0.122	F1
	0.339	0.273	0.244	0.179	P
Random Forests	0.249	0.454	0.150	0.223	R
	0.240	0.334	0.167	0.185	F1
	0.361	0.300	0.241	0.219	P
Extra Trees	0.344	0.464	0.157	0.177	R
	0.352	0.359	0.174	0.183	F1
	NA	NA	0.408	0.133	P
MLkNN	NA	NA	0.068	0.039	R
	NA	NA	0.095	0.0513	F1
	NA	NA	0.010	0.072	P
BPML	NA	NA	0.925	0.303	R
	NA	NA	0.020	0.099	F1

Table 3: Macro Precision, Recall and F1 of flat classification algorithms

performance is discussed in Section 8.
The macro F1 scores are considerably lower than micro F1 scores for all algorithms in the Delicious dataset. While micro-averaging calculates the metrics globally, macro-averaging calculates the metrics for each label and does not take label imbalance into account. This adversely affects the scores in case of datasets with large number of labels, which is the case with Delicious.

7. Feature Selection - RBM

After observing the performance of the different algorithms on the different datasets, we used a neural network based approach (Restricted Boltzmann Machine) for feature selection. In particular, we used a Bernoulli RBM with binary visible units (current feature vector) and binary hidden variables (new feature vector). Parameters are estimated using Stochastic Maximum Likelihood. RBM is a proven method to identify the best features out of a given set of features by exploiting the correlation in the training instances. This is of great use in our particular task, as the reduced dimensionality helps in faster execution of all multilabel classification algorithms

The algorithm used is quadratic in the number of features. Hence, the high dimensional feature vector we deal with in this text classification task proves to be a great challenge in obtaining the optimized feature vector. A parameter choice is the number of hidden variables. We used a $10 : 1$ feature reduction ratio for Delicious and Bookmarks datasets, and used 1000 hidden variables for Citeseer and Reuters datasets.

Once, the reduced feature vectors were obtained, we re-ran the multilabel classification tasks on each of the datasets for all the algorithms. The new predicted labels were used to obtain precision,recall and F1 scores under both micro and macro averaging. (We only discuss interesting observations and inferences. The detailed results are given in the Appendix file). In case of Delicious and Bookmarks dataset, random forest based methods (Random Forest Classifier and Extra Trees Classifier) gave approximately equal values of F1 scores as before, while SVM based methods gave significantly lower values (0.2 - 0.3) of F1 scores. The reduced features were insufficient for SVM to give the same level of performance. In case of Reuters and Citeseer datasets, the F1 scores obtained were far lower than earlier. Thus, the 1000 features obtained using RBM underfit the data. The number of hidden layers chosen is clearly a crucial choice that affects the performance. Due to the limited time and computational resources,

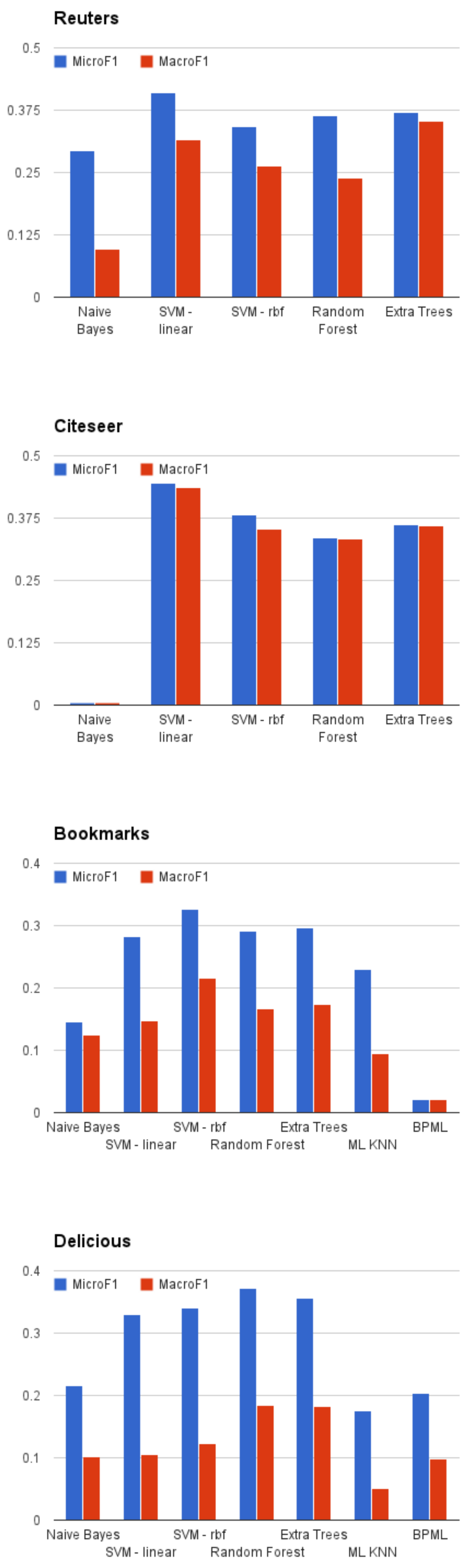

Figure 1: Classifier performance (F1 scores) across datasets

we could only vary this parameter and run the tests for Citeseer dataset. The crux of the observation is that the algorithms give comparable performance in terms of F1 scores at 2000 hidden layers.

Thus, our experiments show that feature selection using RBM can go a long way in both reducing the time complexity and improving the performance by identifying the optimal feature sets. However, this requires careful tuning of the parameters on each of the datasets.

8. Error Analysis

8.1. Text properties of the dataset

*8.1.1. **Bookmarks***

Bookmarks The bookmarks dataset contains metadata for bookmark items such as the URL of the web page, a description of the web page, etc. Below are its 2 example documents that both have 'library' as one of its labels:

```
834
Labels:  library, blog, information, journal,
journals
Document:  access author authors call copyright
editors guidelines index open papers reserved
rights search subscription titles volume

1215
Labels:  library, api, informationretrieval
Document:  download address api apis blog
command data details distance documentation
installation ...
```

As you can see, the first document (834) contains words such as papers, volume, authors, editors etc. This suggests that this document is probably referring to a webpage that contains or indexes a library of papers and their authors, editors etc. On the other hand, the second document (1215) contains words like installation, download, documentation etc, and other tags like api and informationretrieval. This indicates that this document is a library in the API sense, or is a software library.

Such examples suggests that documents in bookmarks that may contain some similar labels could refer to vastly different things, and the relationship between tags that occur together is not very well-defined.

*8.1.2. **Reuters***

This is a well-known dataset consisting of all English newswire articles from Reuters from 1996 to 1997. This dataset contains the topic-codes as the class labels. Below are 2 examples that both contain 'Economic Performance' as one of their label(s):

```
282934
Labels:  Economic Performance
Document headline:  Singapore economic growth
picks up in last quarter.

282958
Labels:  Economic Performance, International
Relations, Domestic Politics, Government...
Document headline:  Taiwanese not so bullish in
Year of the Bull.
```

The document headlines suggest that both documents might be similar in content, and thus contain similar features. Indeed this is true, as we looked at the raw data of the articles, which we do not display here in the interest of space. Such examples indicate that the labels in the Reuters dataset can be linked with features, as there isn't such disparity among documents with similar labels as is in the other Bookmarks dataset.

*8.1.3. **Delicious***

This dataset is extracted from the del.icio.us social bookmarking site on the 1st of April, 2007. It contains textual data of web pages along with their tags. Below are 2 example documents that both have 'cool' as one of their tags:

```
835
Labels:  awesome, artists, audio, awesome,
bands, daily, information, music...
Document:  breathe

1008
Labels:  awesome, photography, pictures,
amazing, art, beauty, cool, culture, ...
Document:  air morning door face lightly ...
```

The first document (835) contains a single word, breathe. Its other labels such as music, audio, bands etc suggest that the webpage contains an audio file. The second document (1008) contains words like air, light, morning, face etc, and other labels such as photography, pictures, art etc. This suggests that this document probably refers to a webpage containing an artistic picture.

Such examples suggest that there is a wide disparity among features of different documents that may have some similar labels.

8.2. Performance of classifiers

- SVM (Linear): From the results obtained, we observe that relatively, SVM performs better than other algorithms in Reuters and Citeseer as compared to the other datasets. On careful investigation, we observe that this is an outcome of the properties of the Reuters and Citeseer datasets. These datasets have higher number of features and fewer labels (as show in Table 1). SVM using a linear kernel is highly effective with high number of features as the data is best separable in such a high dimensional space.
- SVM (RBF): SVM with a rbf kernel performs better than SVM with a linear kernel in Delicious and Bookmarks datasets. These datasets both contain a higher number of labels, but a smaller number of features. Since, SVM with a rbf kernel project the feature vector to an infinite dimensional space, it can separate the data more effectively in such an infinite dimensional space compared to a linear kernel. Moreover, since a one v/s rest implementation of the SVM is used, a separate binary classifier is learnt for each label, which enables better performance compared to the naive baseline.
- RandomForest: Random Forest Classifier gives consistent performance across different datasets, while performing the best in Delicious dataset. Random Forest Classifier, being a decision tree based algorithm, ranks the features by their importances and identifies the best features in the process. Moreover, since it identifies all sets of points with similar feature values, it is best suited to capture instances which have a large number of labels. This explains the superior performance in Delicious dataset which has a cardinality of 19.02.

8.3. Performance in a dataset

To investigate the difference in performance of different classifiers, we studied sample predictions from different classifiers on a particular dataset. We present one such representative example here.

```
<headline>Arab Potash Company 7-month profits
up.</headline>
<dateline>AMMAN 1996-09-05</dateline>
<text> ...  <p>Company officials expect Arab
Potash's profits in 1996 to remain near last
year's record pre-tax profit level of 42
million dinars due to higher production costs
after the government hiked energy costs this
year.</p> ...
<p>The company, a key public shareholding
```

```
company, is 57 percent state-owned but several
Arab states, including Saudi Arabia and Kuwait
have minority holdings.</p> ...  </text>
```

- True Labels: Performance, Accounts/Earnings , Corporate/Industrial
- SVM (Linear) Prediction: Performance, Accounts/Earnings , Corporate/Industrial, Markets/Marketing
- SVM (Rbf) Prediction: Performance, Corporate/Industrial, Science and Technology, Elections
- Random Forest: Performance, Accounts/Earnings , Corporate/Industrial, Comment/Forecast, Capacity/Facilities

Here, as we observe SVM linear does the best prediction. The prediction of SVM with rbf kernel is adversely affected due to the higher weights to words like *goverment, Arab states, public* which results in prediction of *Elections* as a label. Moreover, it fails to capture the label *Accounts/Earnings*. This may be attributed to the low weights in the kernel space to words such as *profits* While random forests does capture all the tags, it results in other extraneous tags. This can be attributed to overfitting in the decision tree which is avoided due to regularization in SVM.

8.4. Flat v/s Hierarchical

Here, we present a representative example that shows the performance of hierarchical classification over flat classification.

```
<headline>Bank of Lebanon weekly CDs raise
four bln pounds.</headline> <dateline>BEIRUT
1996-09-05</dateline> <text> <p>The Bank
of Lebanon issued certificates of deposit
dated September 6 to commercial banks worth
four billion Lebanese pounds compared with 10
billion pounds last week.</p>
<p>The bank sold 60-day CDs worth four billion
compared with five billion last week.  There
were no sales of 45-day CDs this week compared
with five billion last week.</p>
<p>Interest rates on the 45- and 60-day
CDs were 12.75 precent and 13.50 percent
respectively, down from 13 percent and 13.75
percent.</p></text>
```

- True Labels: Markets, Money Markets, Interbank Markets
- Predicted (Flat): Equity Markets, Bond Markets, Energy Markets
- Predicted (Hierarchical): Markets, Money Markets, Bond Markets, Interbank Markets

For this document, the true labels *Money Markets* and *Interbank Markets* are childern of the label *Markets* in the label hierarchy. All classifiers using a flat classification does not take hierarchy into account and treats the training examples as belonging to the classes separately. Hence, the flat classifiers does not learn enough from the training data to capture the fine differences in the words representative of the categories (Money Markets and Bond Markets) that come under the partent category (Markets). However, the hierarchicial classification incorporates this information and hence is able to identify both the parent labels and the children labels for the document.

9. Hierarchical classification

We ran multiple tests with our implemented version of the hierarchical algorithm, which gave us some unique insights into the problem of predicting on a label hierarchy. The variations were:

- results without the final greedy optimization, with just dimension reduction
- dimension of pca for labels
- pca or kpca (while reducing label-space)
- and-tree or or-tree (maintain the constraint that if child is predicted then all or one of the parents is predicted)
- predict L labels, where L varies from 1 to 30

9.1. Effect of dimension reduction

We first look at how dimension reduction helps in classification. Remember that at the preprocessing stage, we project the label-space (not the feature space) to a lower dimension and learn regressors in that dimension and project the answers back to the original dimension. Using this preprocessing alone, i.e. using pca to reduce the dimensions of the label-space, we saw market differences in how different datasets behave. For rcv1/reuters, we got a significant F1 of 74% using this method, while the F1 arrived at from previous experiments using SVM and others hovered around 40%. While this result was baffling, the equivalent method for citeseer and others didn't see any significant improvement and we got comparable F1 scores. The performance was either comparable or less that rbf. We assume this is due to the reason that using all the dimensions in rcv1 significantly overfits the data. Since rcv1 has 3-4 labels per data-point, labels at the leaf might have fewer examples compared to labels near the root, leading to an overfit when we are doing a flat classification. Also the prediction performance is impaired as structural dependencies among labels are not utilized in the learning process. This is not a big deal in case of other datasets since bookmarks and delicious don't have a structure in their labelsets and for citeseer, we have around 1 label per document on an average, mitigating the overfitting due to discrepancy in data per label.

9.1.1. Effect of pca dimension

Different datasets behave differently with respect to dimensions. For example, on reuters data, the highest F1 remains the same and increases only mildly as we go from 30 to 100 dimensions (73.67-74.19%), while for citeseer, the change is significant (36.95-42.15%). We looked into the pca matrix, and this discrepancy turned out to be due to the distinction between distribution of variance in the two matrices. While 90% of variance in the reuters dataset is confined to 20-25 dimensions, in citeseer, almost all of the 83 possible dimensions have close weights. There is no significant dimension which hogs a large portion of the variance, leading to almost uniform distribution through out the dimensions, calling for a higher number of dimensions to give better results. This is possibly the outcome of the fact that even though both the datasets boast of comparable number of labels (83 for citeseer and 101 for rcv1), the average label per document is 1.44 in citeseer as compared to 3.18 in rcv1.

Consequently, the citeseer label-set that is extracted from the training data has a lot of variance, there being just $\tilde{1}$ label per item, where as there is greater possibility of overlap in rcv1 where we have 3-4 labels per data-point. So this gives us a way to customize the dimension for different kinds of datasets. Also note that since pca was done on the label set of the training data (unlike kpca, which works on the label-hierarchy-tree), this dimension calculation would be heavily training set dependent.

9.1.2. Choice of pca/kpca

The experiments in this front gave us remarkably striking results. While using kpca gave us better results than pca when we were using citeseer dataset (41% compared to 37%), the trend was completely the opposite in the case of reuters/rcv1 (42% compared to 74%). To investigate this, we looked into the label hierarchy structure of both datasets. For citeseer, out of 83 labels, only 17 labels are parents, with 4 of them having 3 children each and all else but 1 having 4 children, with the final node having 6 children. Evidently, there are too few branches in the tree and even then, the branching is extremely structured with minimal variation.

The rcv1 hierarchy, on the other hand, has 35 odd parent nodes, with number of children varying from 1 to a maximum of 23, showing evidence of a significant variance in tree structure. Consequently, while kpca on the citeseer hierarchy can lead to significant dimensions which have the maximum weight variance, the same won't be true due to significant variance on different fronts for the rcv1 dataset and the resultant dimensions would have close to equal contribution. In fact, a look into their eigen-values confirms this, as citeseer takes only 16 significant eigen-vectors while the same number for rcv1 is 26. At the same time, as we explained in the previous subsection, due to having 3-4 labels per data point on an average, the variance of label-set (not the label-tree, which is what we were concerned with in kpca), is concentrated in few eigenvectors while it's a large number for citeseer (e.g. the 10th eigen vector reduces to 10% value of the highest in rcv1, while for citeseer, it's the 70th). Both these effets simultaneously lead to such drastic effects.

9.2. Number of labels predicted

Remember that the post processing on trees gives out L possible labels. As we alluded to before, the post-processing step is a greedy solution for a fractional knapsack like optimization. As such, effectively it finds out clusters or subgraphs in the hierarchy tree with highest mean weight. So, if it's asked to provide 5 labels, it might out put 3 labels that belong to 1 sub-tree and 2 that belong to another.

We would now like to draw attention to the resultant micro-f1 graphs for 8 different combinations of experiment: (rcv1 or citeseer), (kpca or pca), and (tree or no tree i.e. post processing or no post-processing). The effect of pca/kpca has already been dealt with in detail in the previous section. Of special interest are the kinks in the tree. Notice that the F1 scores goes up for a while and them drops. While the shape looks the same for both the case with trees and with no trees, they have a different origin story. When no tree is used, the F1 is low at first because even though we have a very high precision of predicting the highly likely labels, we suffer on recall. Notice that the kink for tree-less citeseer is around 1 or 2 while for that of rcv1 is at 3-4; this is because they have on an average that number of labels per data point. So, while we are predicting less than the average labels, we are sacrificing recall, and when we go above it, we begin penalizing the precision. The kink in the tree-related citeseer graph is however at 3-4 (look at the very evident green lines, which have kinks both at 3 and 4). It's primarily because the hierarchical algorithm has a tendency to predict whole branches at once. It's somewhat explainable by the assumption that labels in a particular branch would all be significant and secondly, the algorithm explicitly enforces a condition for predicting a parent if a child is also predicted. If this condition has a strong enough value, then the whole branch is predicted.

However, consider a case when we are asking for 2 labels and a whole branch of 4 is predicted. The fractional knapsack like algorithm in that case outputs no results, leading to zero precision and recall. Since citeseer has branch lengths of 3 and 4 only (described in a previous subsection), only after going over that threshold, we begin getting significant F1's, which were very low due to 0 predictions, leading to a kink in the graph. For reuters dataset, that number seems close to 10. Also notice that due to this effect, even though the post processing based on a tree gives poor results at first, it begins outperforming the un-processed results soon after that. So, if somebody is in a mood to use this algorithm, they should be aware of this aspect and choose whether to run the post-processing step or not based on information about the label hierarchy for the dataset.

9.3. Effect of tree

We mentioned how the post processing on tree might help increase performance after a certain point in the previous section. Another aspect of the algorithm is the choice of either the OR-tree or AND-tree. However, they are relevant for only DAG structured hierarchies. Our hierarchies as tree structured and as such, both are equivalent.

10. Conclusion and Future Work

In this project we tackled multiple issues related to multilabel text classification. We experimented on different fronts, to wit, different data sets, algorithms, feature transformations and incorporation of hierarchical label space information. We talked about the peculiarities in the results we got and tried to explain them through a thorough analysis of the underlying algorithm, especially in the hierarchical algorithm part, which we implemented from scratch. The insights we gained would further guide us towards choosing certain algorithms and their parameters that cater to certain aspects of the dataset we would be dealing with. Specifically, we have provided a detailed analysis of how to choose various parameters of the hierarchical multilabel predictor and which parts of this algorithm to use. We modified the original algorithm that ran only KPCA to incorporate pca information and talked in details about how it affects the performance; it's to be noted that using this small adjustment, we got the highest F1 for rcv1 dataset compared to all the other standard methods. We also saw that it doesn't necessarily work in case of all other datasets, and we pointed out some aspects of a dataset, especially rcv1, for which it might work wonders.

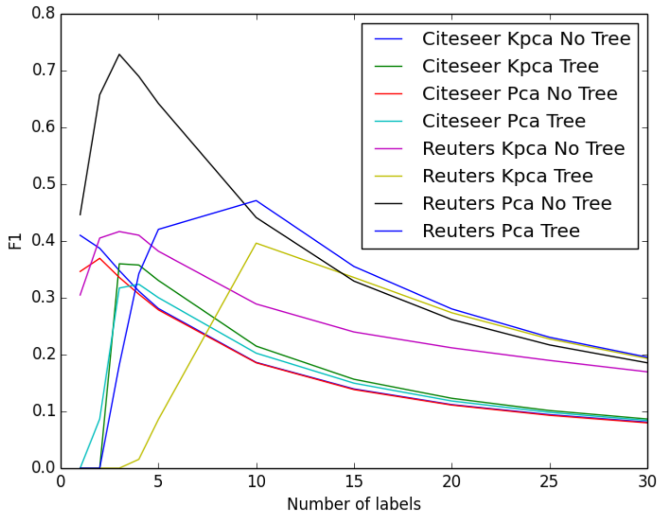

Figure 2: CSSAG performance across K-best labels

As far as future work is concerned, we have begun looking into multiple hidden layer enforced deep learning algorithms. In particular, we have already implemented DBN from scratch (included in submission for reference) and we hope to run further experiments on it in the future. Owing to the large scope of the present project and considering that running of deep networks takes insane amount of time and fine-tuning, we relegate that task to a future endeavor.

11. Acknowledgements

We would like to acknowledge the Stanford Linguistics department for generously making available to us the Reuters dataset in its raw format. We also obtained two of our datasets, Bookmarks and Delicious from Mulan, a Java library for Multi-Label learning. We acquired the Citeseer data published by Siddharth Gopal of CMU. We used the scikit python library version 0.14 implementation of Naive Bayes, SVM, Extra Trees and Random Forest algorithms.

12. References

[1] Charu C Agarwal, ChengXiang Zhai, A Survey of Text Classification Algorithms

[2] Grigorios Tsoumakas, Ioannis Katakis, Multi-Label Classification: An overview

[3] Wei Bi, James T Kwok, Multilabel Classification on Tree and DAG structured Hierarchies

[4] Barutcuoglu, Z. and Troyanskaya, O.G. Hierarchical multi-label prediction of gene function

[5] Li, T and Ogihara, M, Detecting emotion in music

[6] Boutell, M R., Luo, J., Shen, X. and Brown, C.M., Learning multi-label scene classification

[7] Zhang, M-L and Zhou, Z.H , A k Nearest Neighbor Based Algorithm for Multilabel Classification

[8] Min-Ling Zhang and Zhi-Hua Zhou, Senior Member, IEEE, Multi-Label Neural Networks with Applications to Functional Genomics and Text Categorization

[9] Zheng, Z., Wu, X., Srihari, R.: Feature selection for text categorization on imbalanced data. SIGKDD Explorations Newsletter, 6(1), pp. 80–89 (2004)

[10] Tsoumakas, G., Vlahavas I.: Random k-Labelsets: An Ensemble Method for Multilabel Classification. In Proceedings of the 18th European Conference on Machine Learning, pp. 406–417, Springer Verlag, LNAI 4701, Warsaw, Poland (2007)

[11] Schapire, E., Yoram ,S.: BoosTexter: A boosting-based system for text categorization. Machine Learning (39) pp.135–168 (2000)